Collections, Part Three

Lexicon

Lexicon

- A `Lexicon` is a container that stores a collection of words.
- The `Lexicon` is designed to answer the following question efficiently:

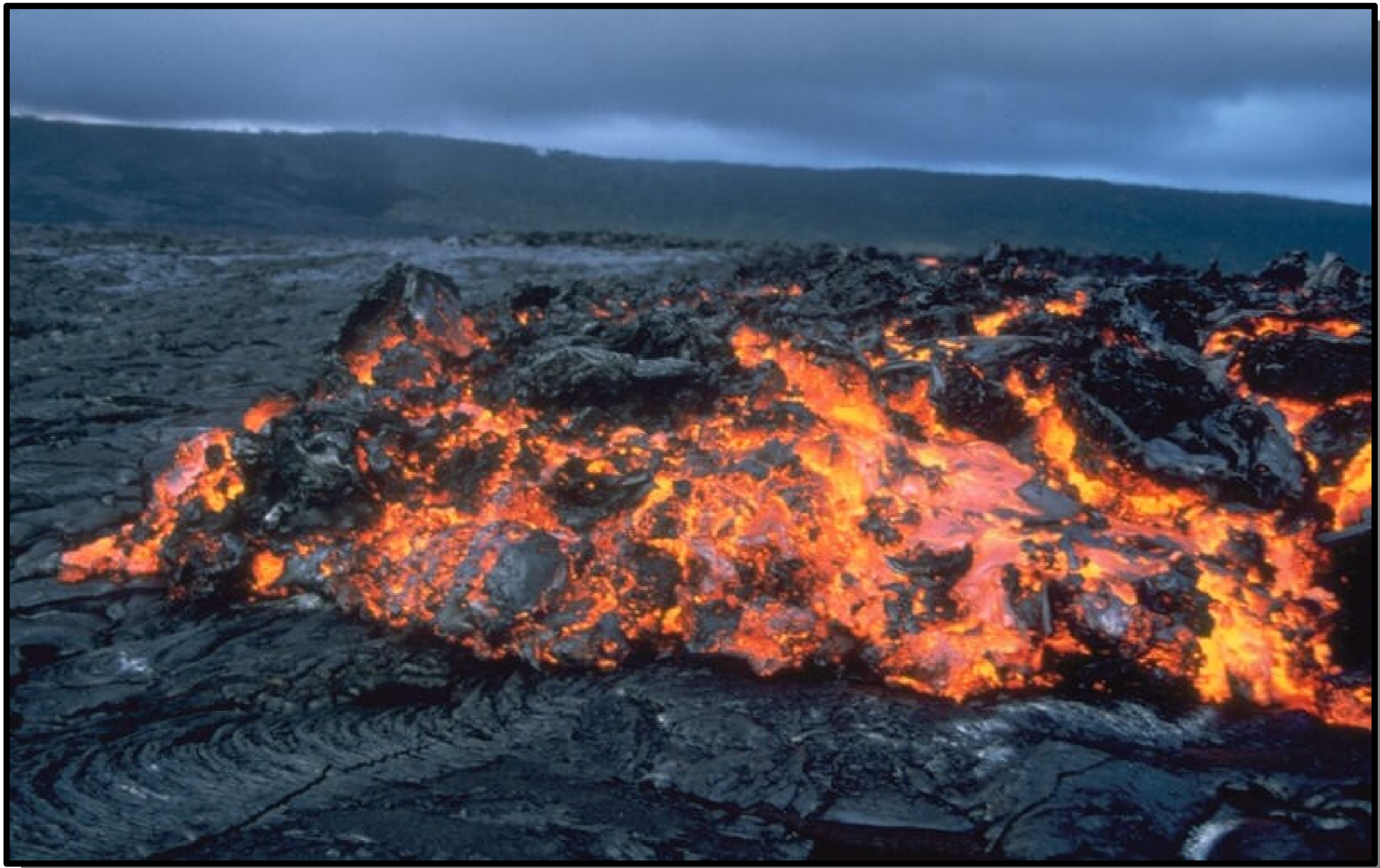
Given a word, is it contained in the `Lexicon`?

- The Lexicon does *not* support access by index. You can't, for example, ask what the 137th English word is.
- However, it *does* support questions of the form "does this word exist?" or "do any words have this as a prefix?"

Tautonyms

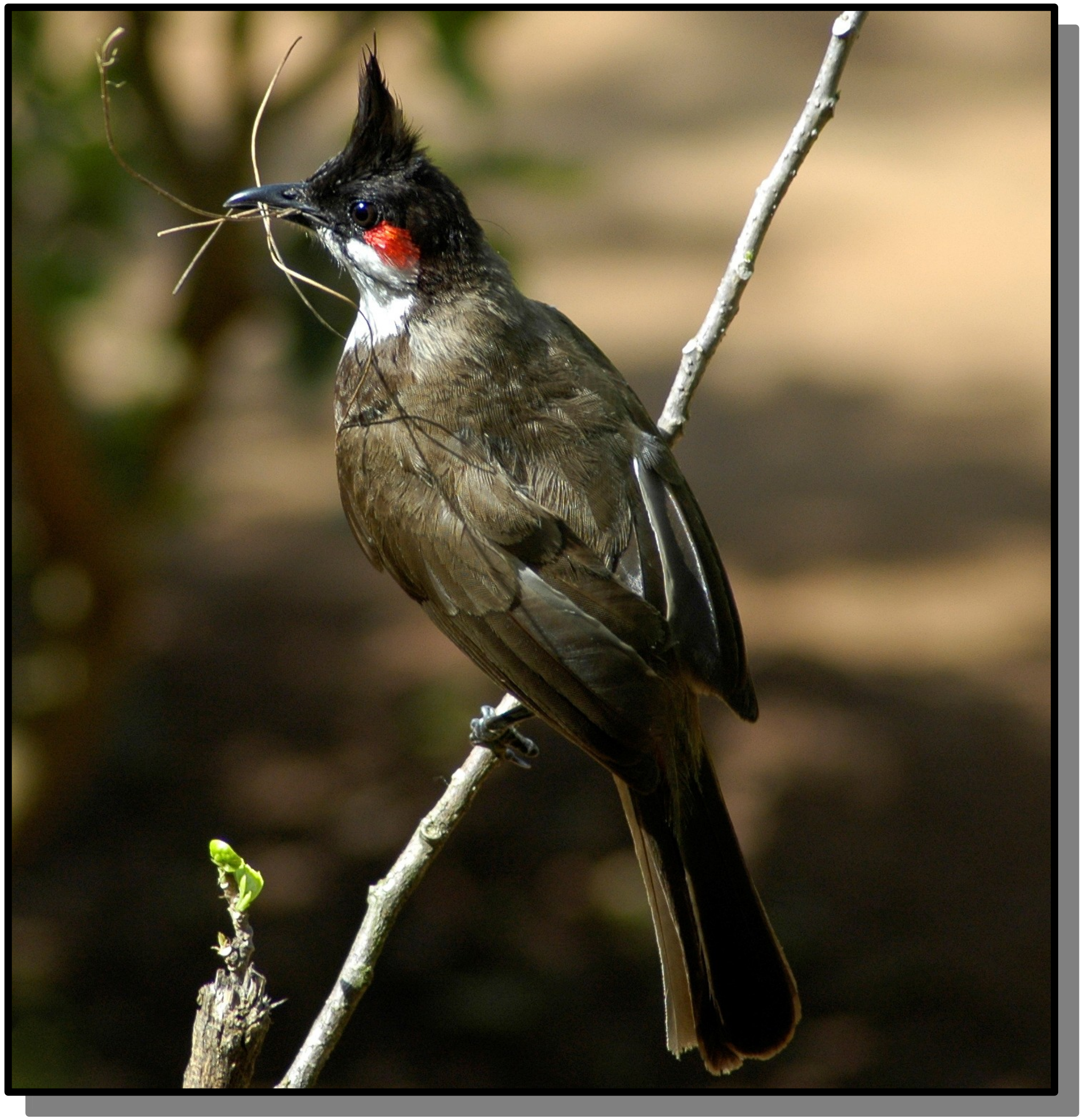
- A ***tautonym*** is a word formed by repeating the same string twice.
  - For example: murmur, couscous, papa, etc.
- What English words are tautonyms?

Some Aa



http://upload.wikimedia.org/wikipedia/commons/f/f1/Aa_large.jpg

One Bulbul



More than One Caracara

http://www.greglasley.net/images/CO/Crested-Caracara-F3.jpg

Introducing the Dikdik

And a Music Recommendation

Time-Out for Announcements!

Assignment 2

- Assignment 2 (Fun with Collections) goes out today. It's due a week from this Monday.
    - Explore mathematical models of crystal formation!
    - Build a program that (almost) always wins at *Hangman*.
- We've provided a suggested timetable for completing this assignment on the front page of the handout. Aim to stick to this timeline; you've got plenty of time to complete things if you start early.
- ***You must complete this assignment individually***. Working in pairs is not permitted yet.

YEAH Hours

- We’ll be holding YEAH (Your Early Assignment Help) hours for this assignment today at 3:30PM in Shriram 104.
- Can’t make it? No worries! The slides will be up on the course website.

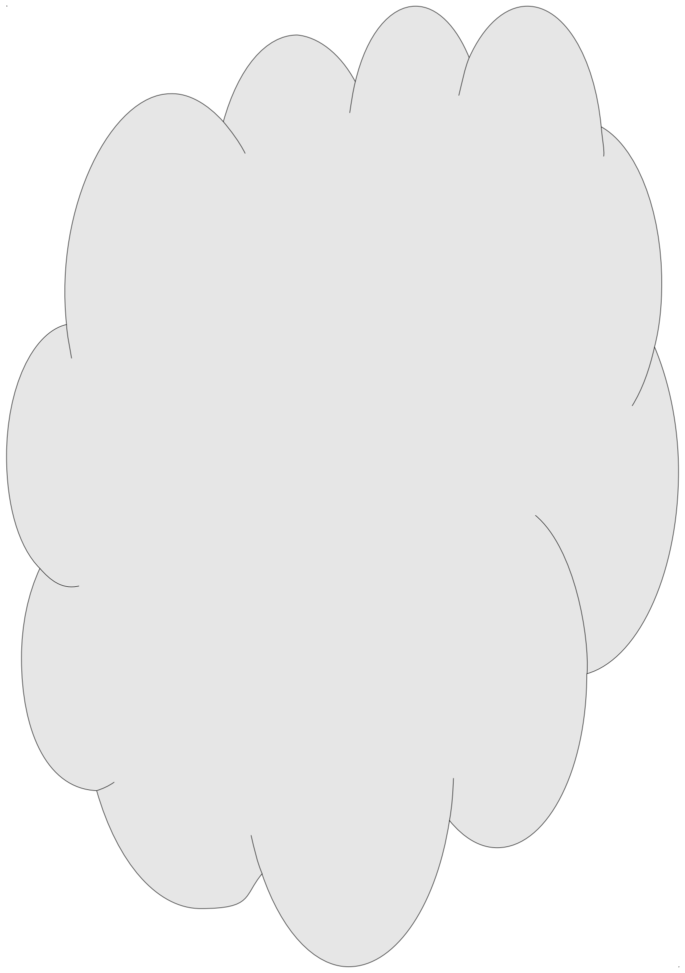
LaIR Closure

- The LaIR will be closed on Sunday in observance of Dr. Martin Luther King, Jr. Day.
- The LaIR will, however, be open on Monday during the usual 7PM – 11PM time slot.

Set

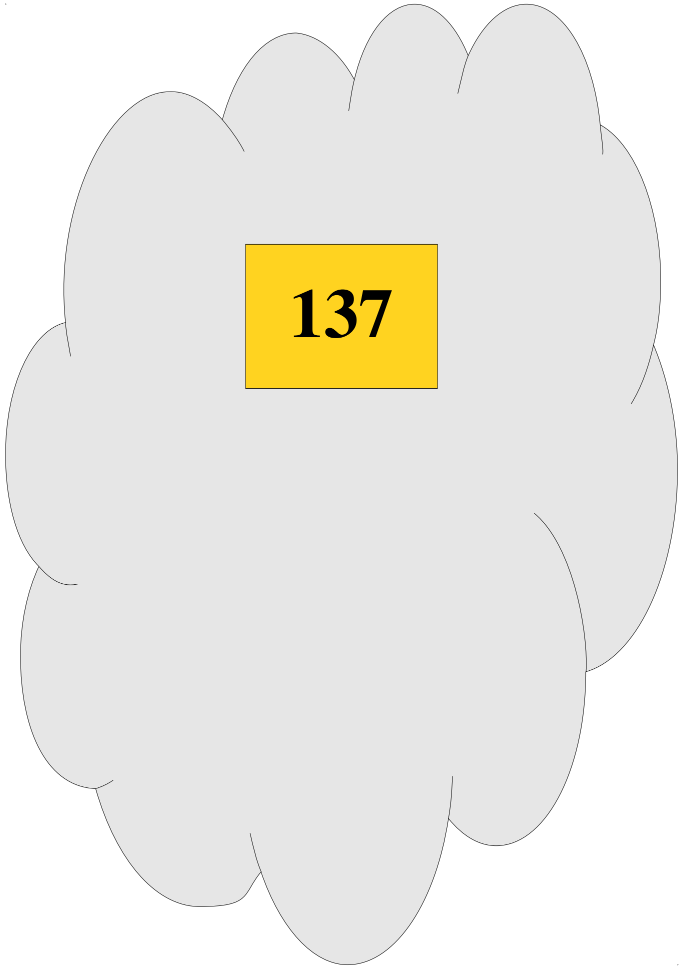
Set

- The `Set` represents an unordered collection of distinct elements.
- Elements can be added and removed, and you can check whether or not an element exists.



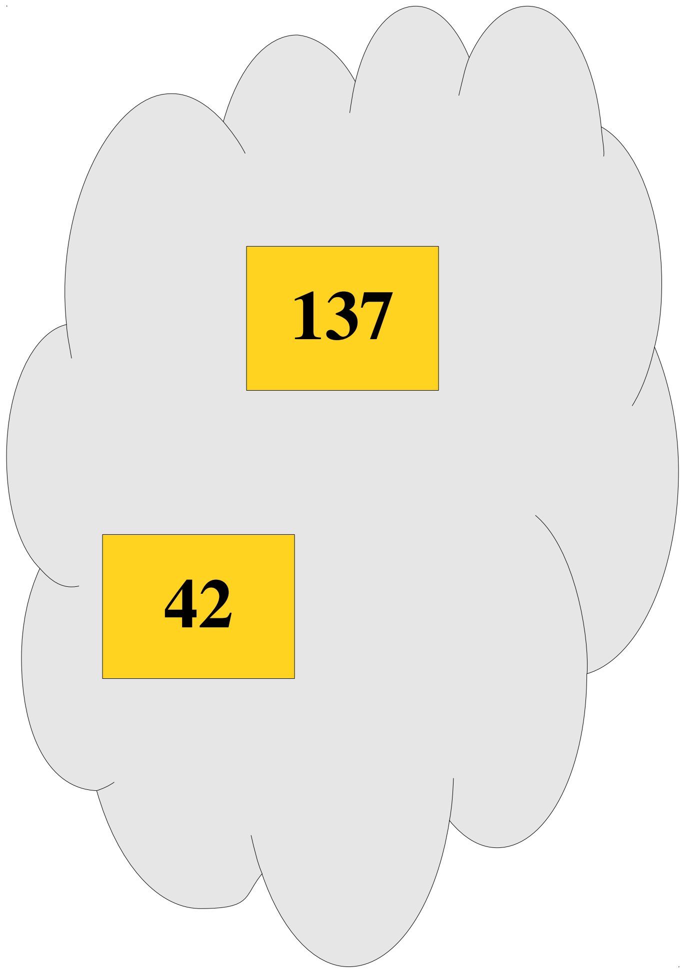
Set

- The `Set` represents an unordered collection of distinct elements.
- Elements can be added and removed, and you can check whether or not an element exists.



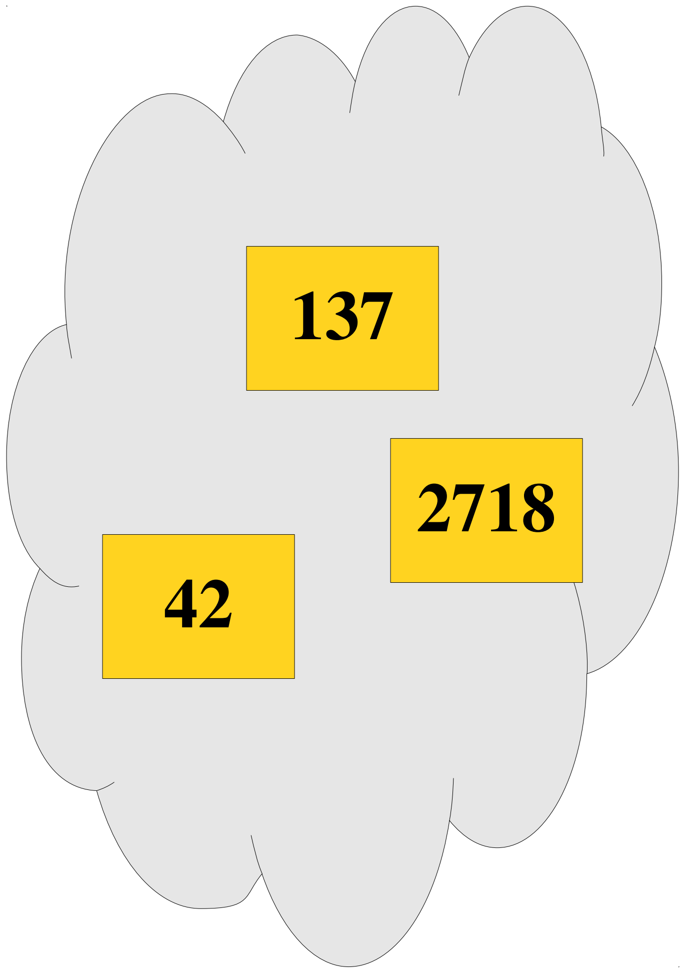
Set

- The `Set` represents an unordered collection of distinct elements.
- Elements can be added and removed, and you can check whether or not an element exists.



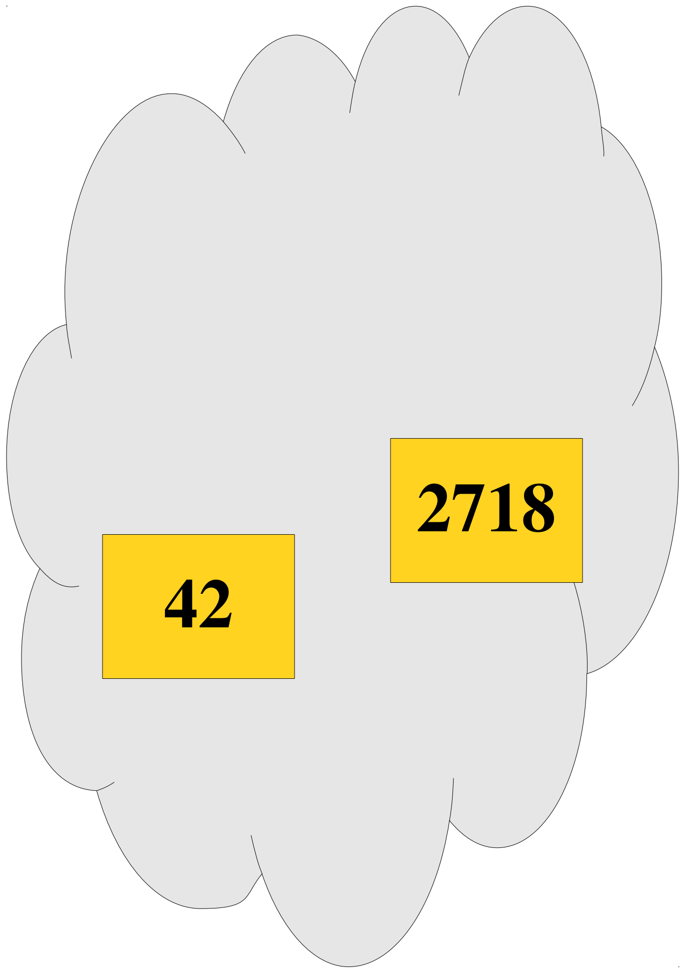
Set

- The **Set** represents an unordered collection of distinct elements.
- Elements can be added and removed, and you can check whether or not an element exists.



Set

- The `Set` represents an unordered collection of distinct elements.
- Elements can be added and removed, and you can check whether or not an element exists.



Operations on Sets

- You can add a value to a set by writing

```
set += value;
```

- You can remove a value from a set by writing

```
set -= value;
```

- You can check if a value exists by writing

```
set.contains(value)
```

- Many more operations are available (union, intersection, difference, subset, etc.), so be sure to check the documentation.

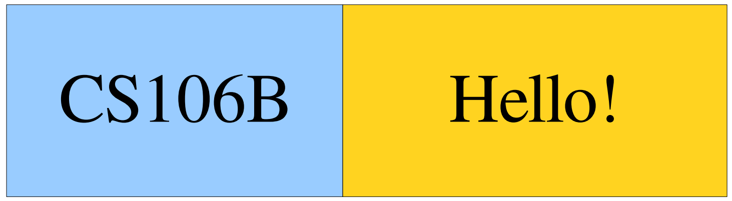
Map

`Map`

- The `Map` class represents a set of key/value pairs.
- Each key is associated with a unique value.
- Given a key, can look up the associated value.

Map

- The **Map** class represents a set of key/value pairs.
- Each key is associated with a unique value.
- Given a key, can look up the associated value.



`Map`

- The **`Map`** class represents a set of key/value pairs.
- Each key is associated with a unique value.
- Given a key, can look up the associated value.

CS106B	Hello!
Dikdik	Cute!

`Map`

- The `Map` class represents a set of key/value pairs.
- Each key is associated with a unique value.
- Given a key, can look up the associated value.

CS106B	Hello!
Dikdik	Cute!
This Slide	Self Referential

`Map`

- The `Map` class represents a set of key/value pairs.
- Each key is associated with a unique value.
- Given a key, can look up the associated value.

CS106B	Hello!
Dikdik	**Very** Cute!
This Slide	Self Referential

Using the `Map`

- You can create a map by writing

```
Map<KeyType, ValueType>  map;
```

- You can add or change a key/value pair by writing

```
map[key] = value;
```

  If the key doesn't already exist, it is added.

- You can read the value associated with a key by writing

```
map[key]
```

  If the key doesn't exist, it is added and associated with a default value.

- You can check whether a key exists by calling

```
map.containsKey(key)
```

Map Autoinsertion

```
Map<string, int> freqMap;
while (true) {
    string text = getLine("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}
```

Map Autoinsertion

```
Map<string, int> freqMap;
while (true) {
    string text = getLine("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}
```

Map Autoinsertion

```
Map<string, int> freqMap;
while (true) {
    string text = getLine("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}
```

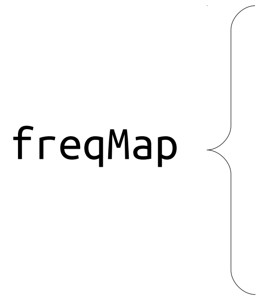
freqMap

Map Autoinsertion

```
Map<string, int> freqMap;
while (true) {
    string text = getLine("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}
```

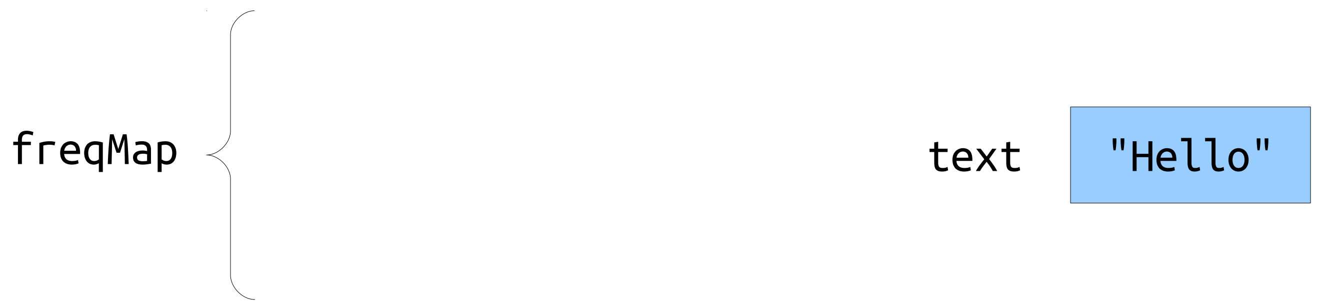
Map Autoinsertion

```
Map<string, int> freqMap;
while (true) {
    string text = getLine("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}
```



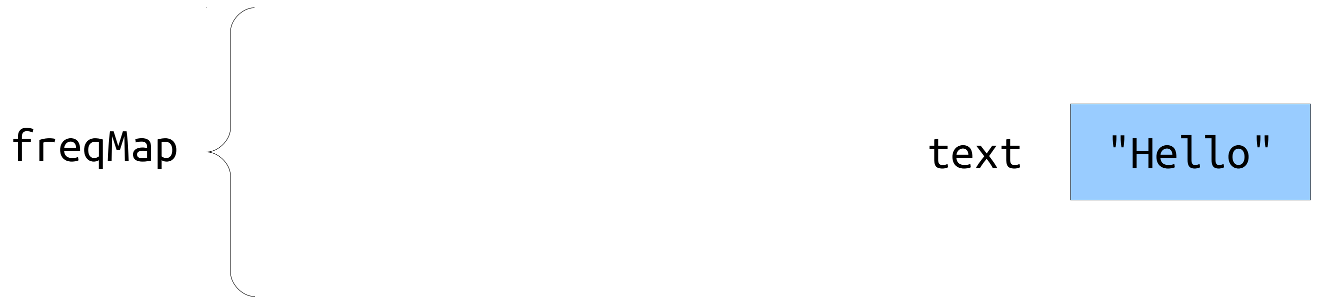
Map Autoinsertion

```
Map<string, int> freqMap;
while (true) {
    string text = getLine("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}
```



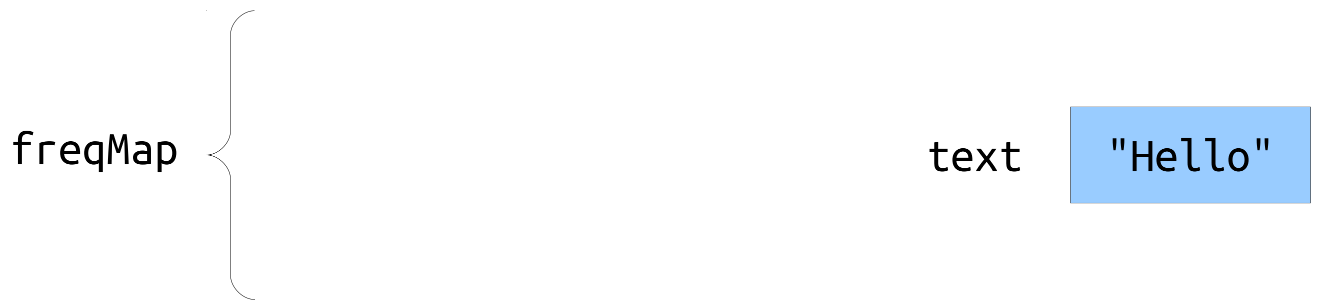
Map Autoinsertion

```
Map<string, int> freqMap;
while (true) {
    string text = getLine("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}
```



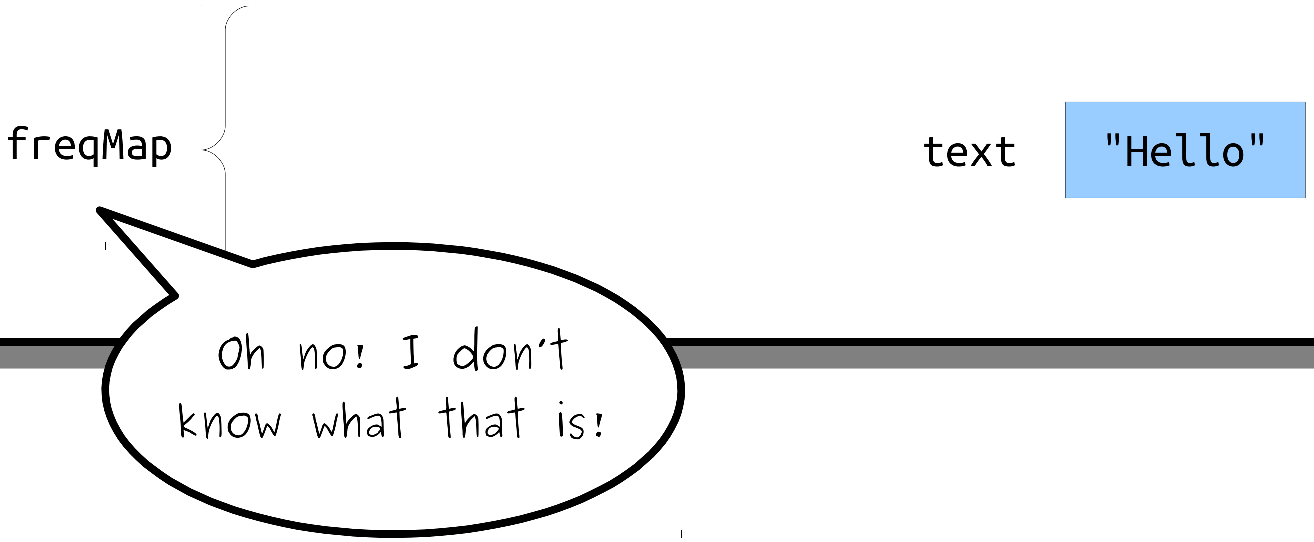
Map Autoinsertion

```
Map<string, int> freqMap;
while (true) {
    string text = getLine("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}
```



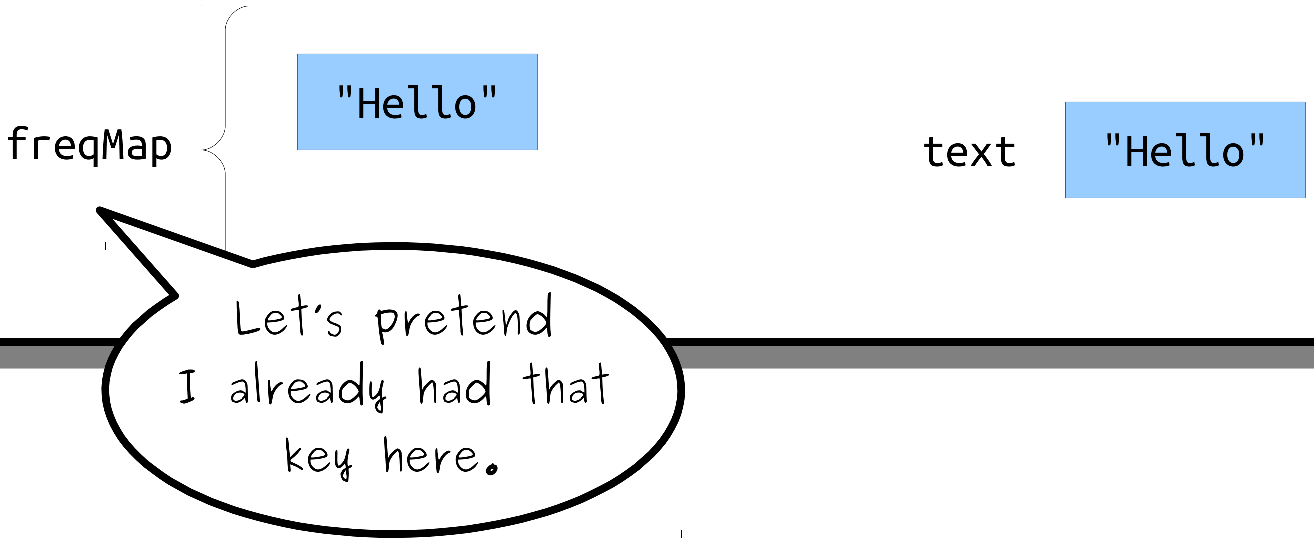
Map Autoinsertion

```
Map<string, int> freqMap;
while (true) {
    string text = getLine("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}
```



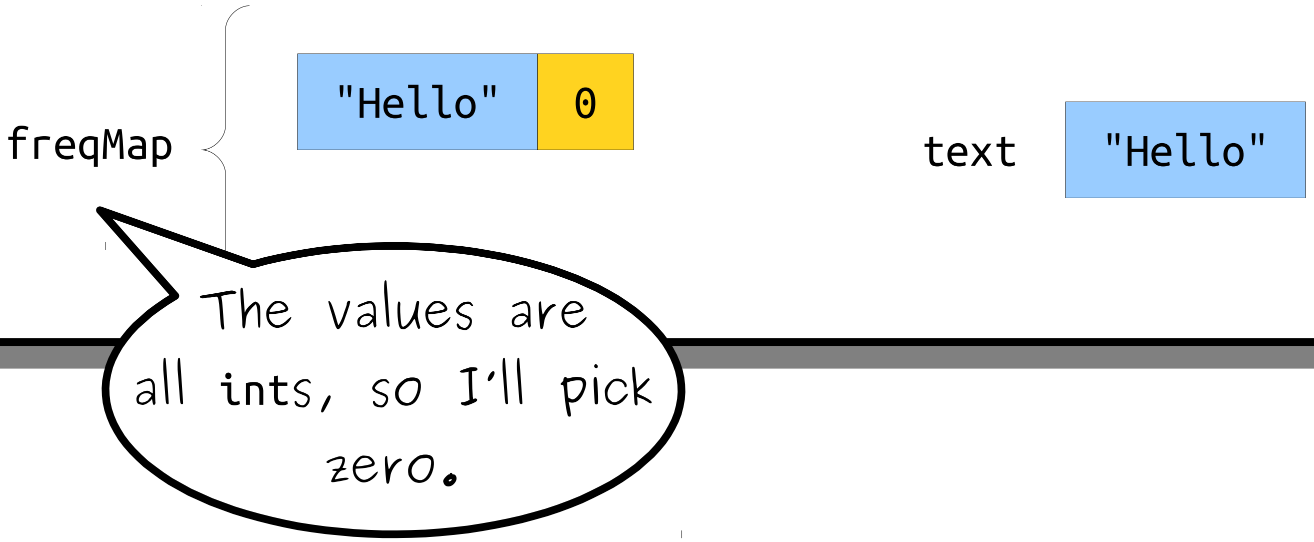
Map Autoinsertion

```
Map<string, int> freqMap;
while (true) {
    string text = getLine("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}
```



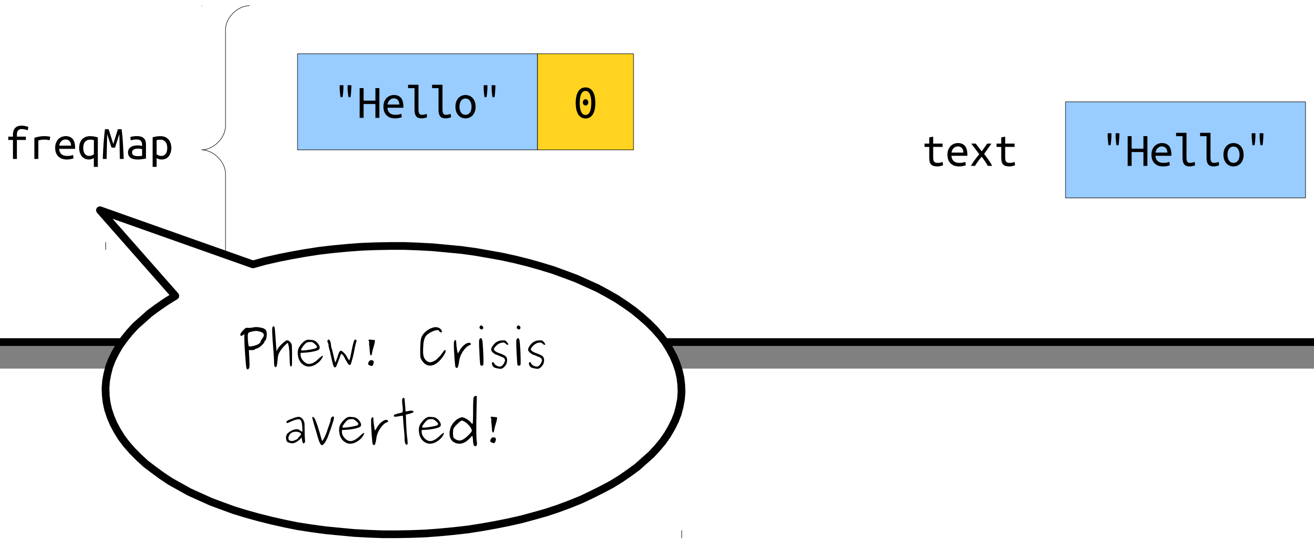
Map Autoinsertion

```
Map<string, int> freqMap;
while (true) {
    string text = getLine("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}
```



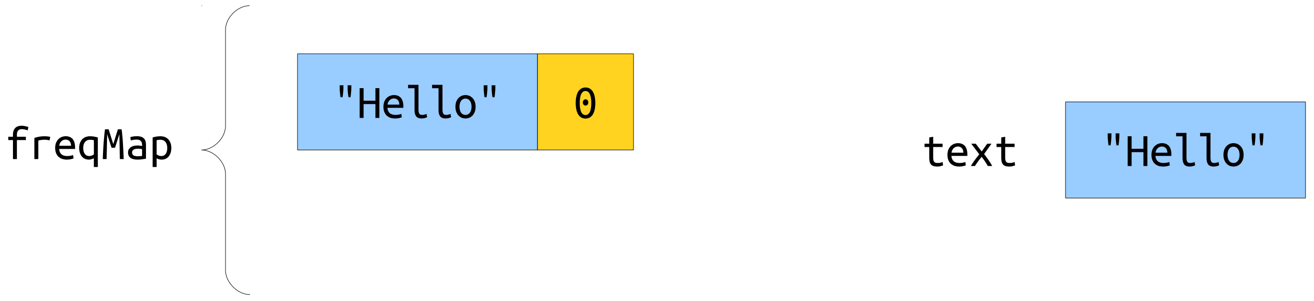
Map Autoinsertion

```
Map<string, int> freqMap;
while (true) {
    string text = getLine("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}
```



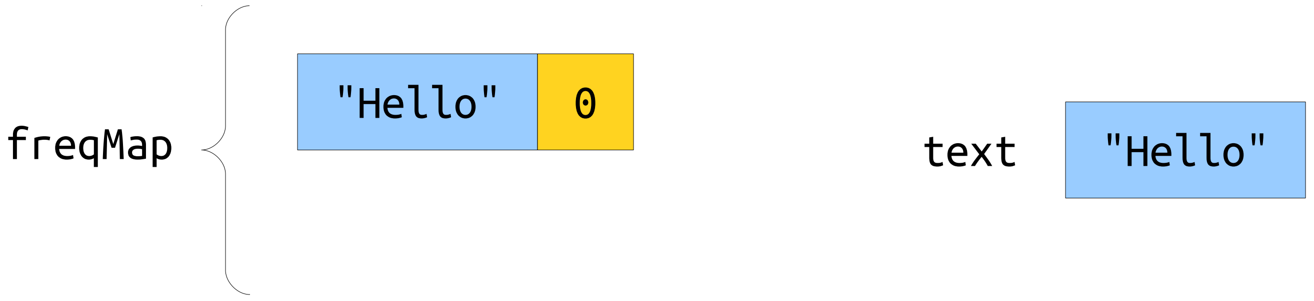
Map Autoinsertion

```
Map<string, int> freqMap;
while (true) {
    string text = getLine("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}
```



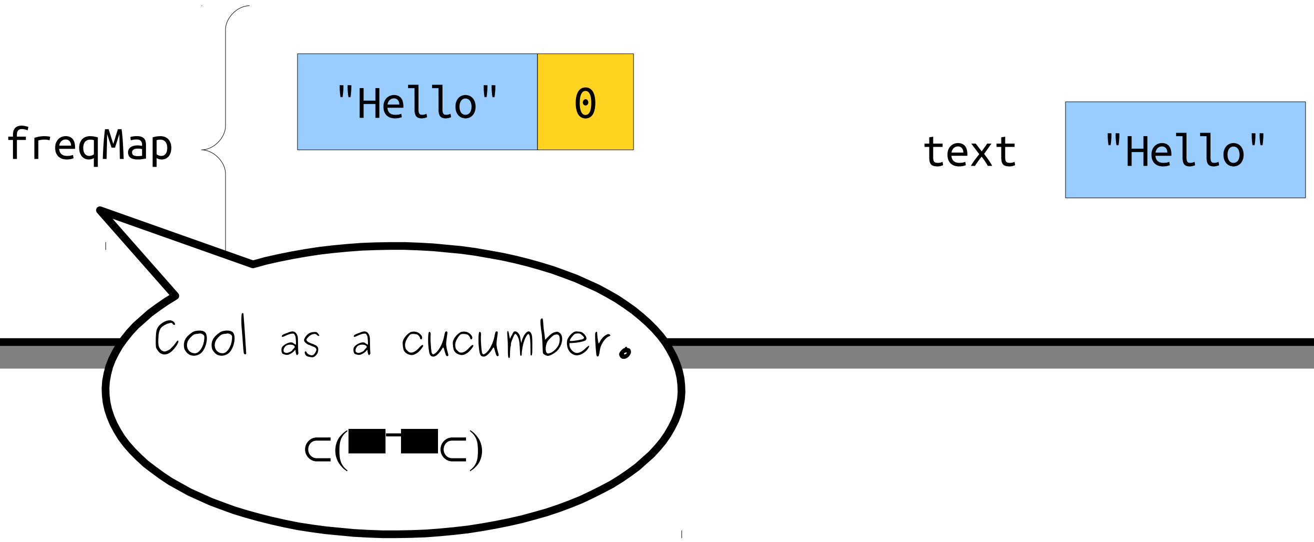
Map Autoinsertion

```
Map<string, int> freqMap;
while (true) {
    string text = getLine("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}
```



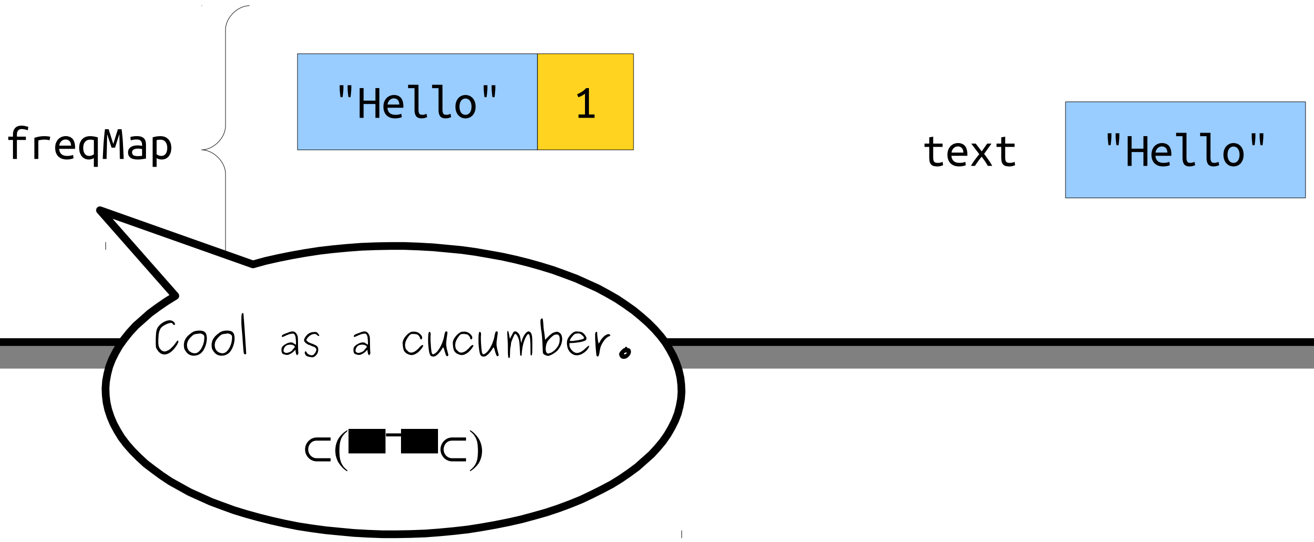
Map Autoinsertion

```
Map<string, int> freqMap;
while (true) {
    string text = getLine("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}
```



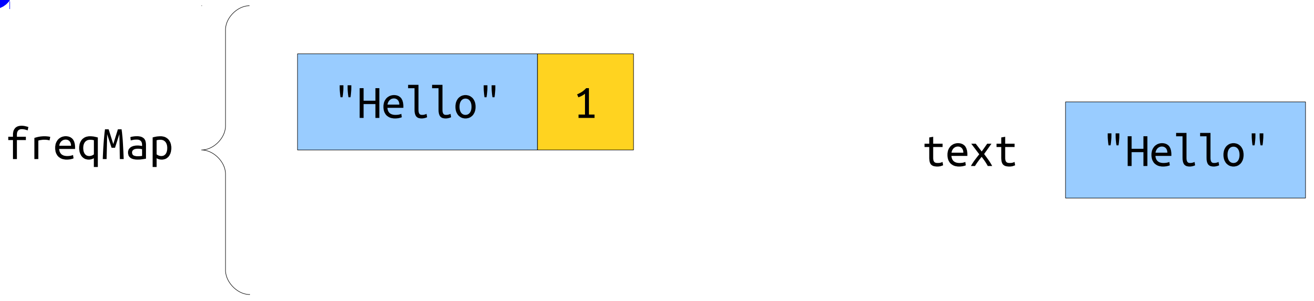
Map Autoinsertion

```
Map<string, int> freqMap;
while (true) {
    string text = getLine("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}
```



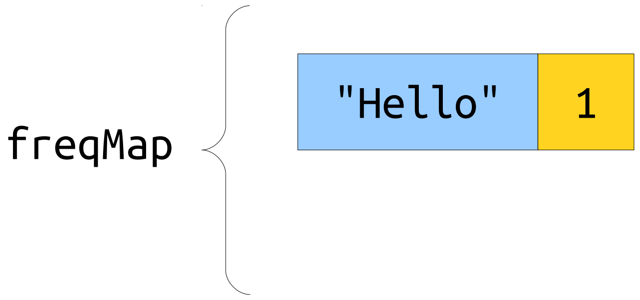
Map Autoinsertion

```
Map<string, int> freqMap;
while (true) {
    string text = getLine("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}
```



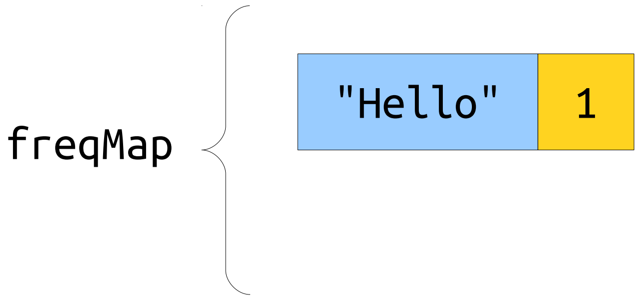
Map Autoinsertion

```
Map<string, int> freqMap;
while (true) {
    string text = getLine("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}
```



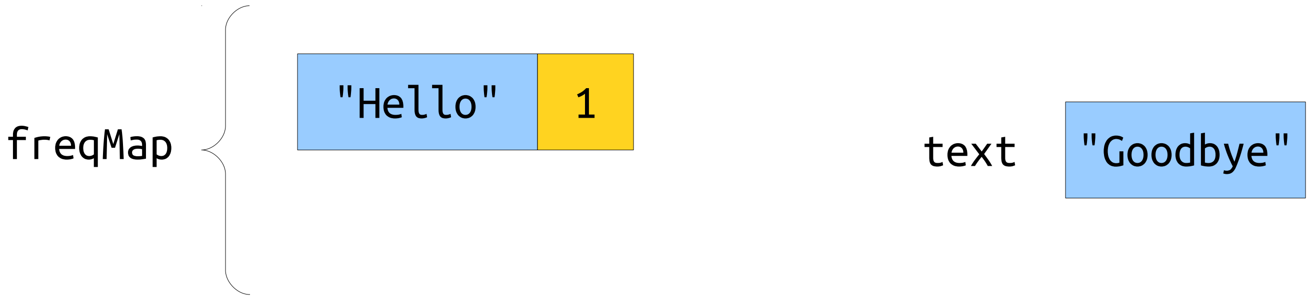
Map Autoinsertion

```
Map<string, int> freqMap;
while (true) {
    string text = getLine("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}
```



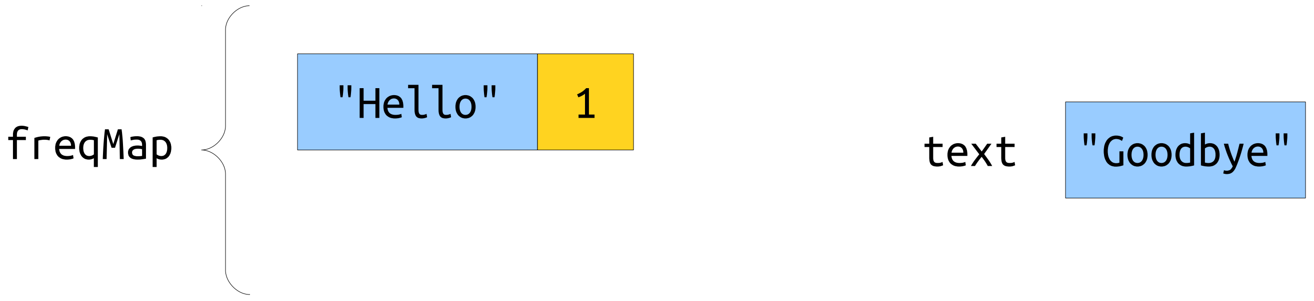
Map Autoinsertion

```
Map<string, int> freqMap;
while (true) {
    string text = getLine("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}
```



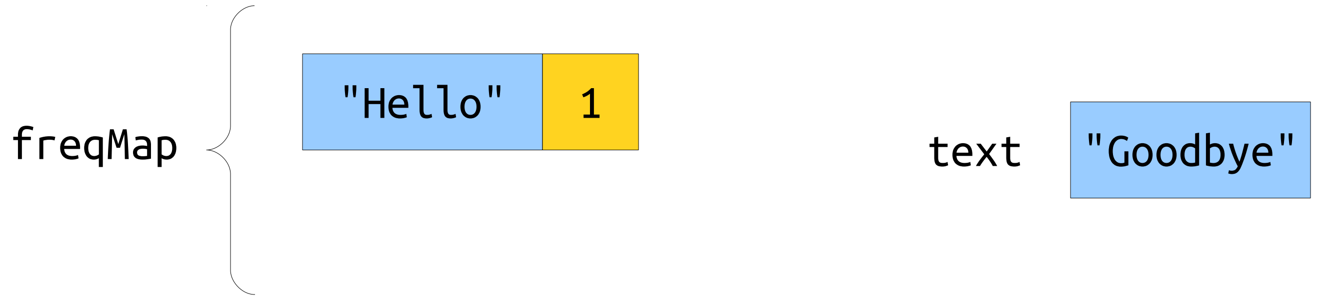
Map Autoinsertion

```
Map<string, int> freqMap;
while (true) {
    string text = getLine("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}
```



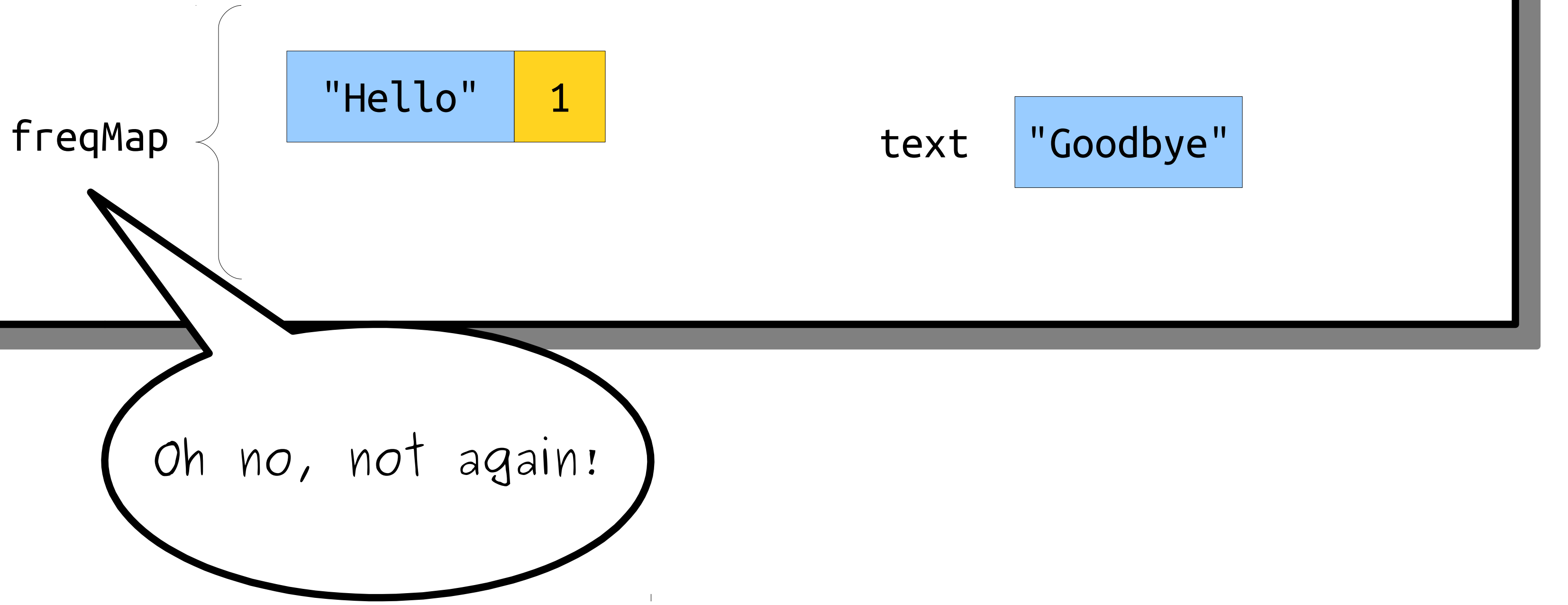
Map Autoinsertion

```
Map<string, int> freqMap;
while (true) {
    string text = getLine("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}
```



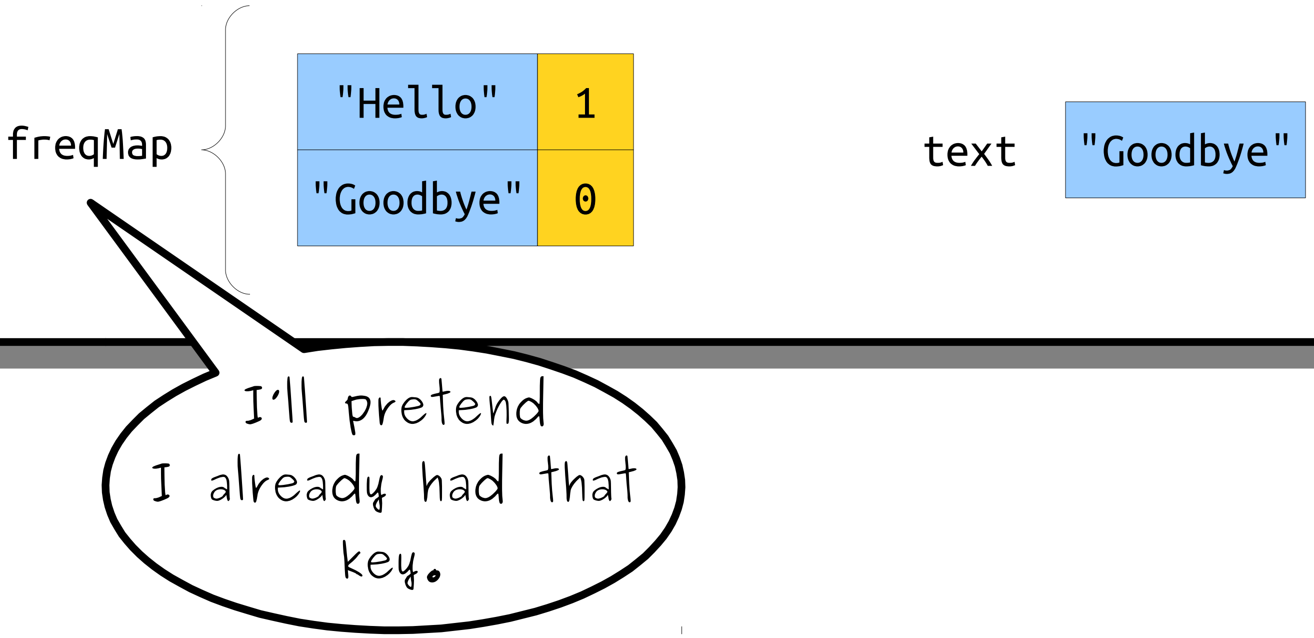
Map Autoinsertion

```
Map<string, int> freqMap;
while (true) {
    string text = getLine("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}
```



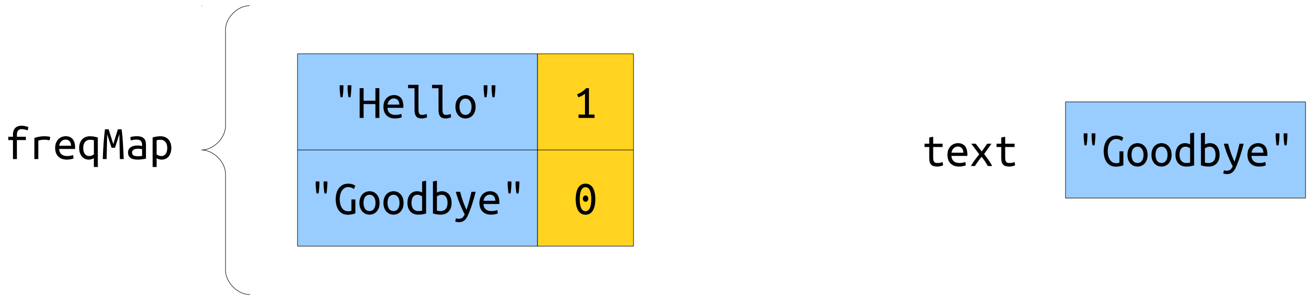
Map Autoinsertion

```
Map<string, int> freqMap;
while (true) {
    string text = getLine("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}
```



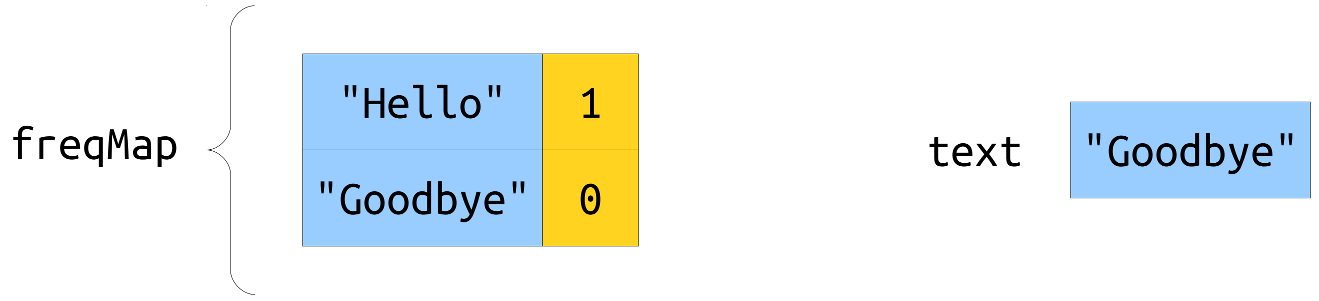
Map Autoinsertion

```
Map<string, int> freqMap;
while (true) {
    string text = getLine("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}
```



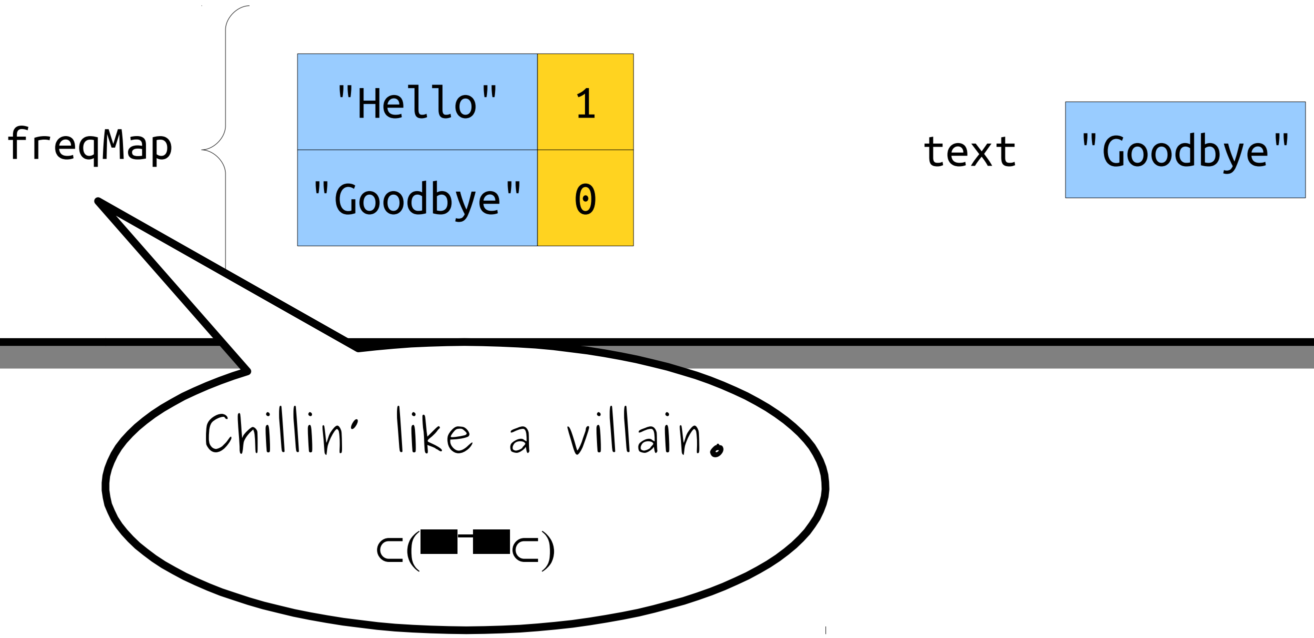
Map Autoinsertion

```
Map<string, int> freqMap;
while (true) {
    string text = getLine("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}
```



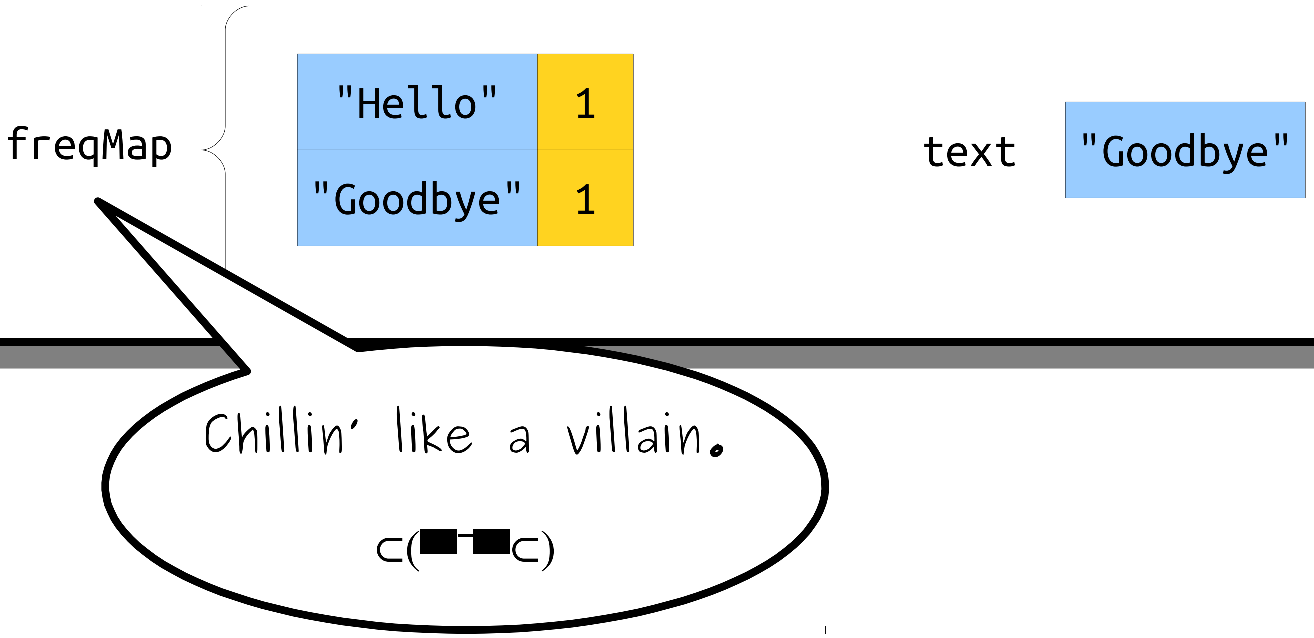
Map Autoinsertion

```
Map<string, int> freqMap;
while (true) {
    string text = getLine("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}
```



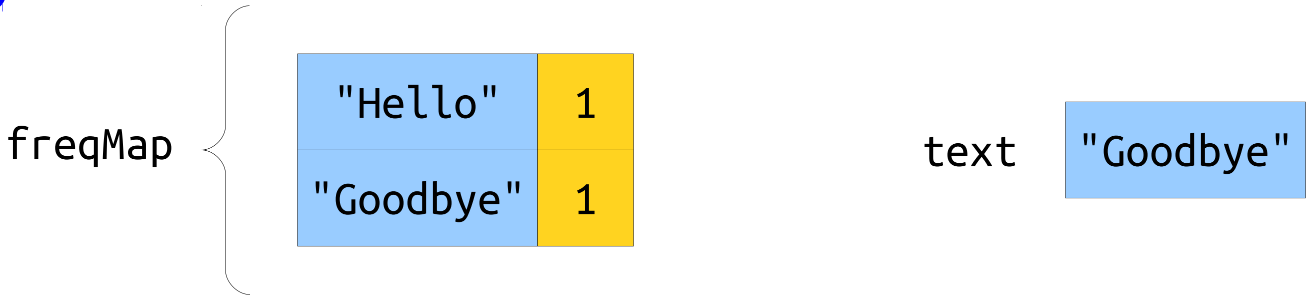
Map Autoinsertion

```
Map<string, int> freqMap;
while (true) {
    string text = getLine("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}
```



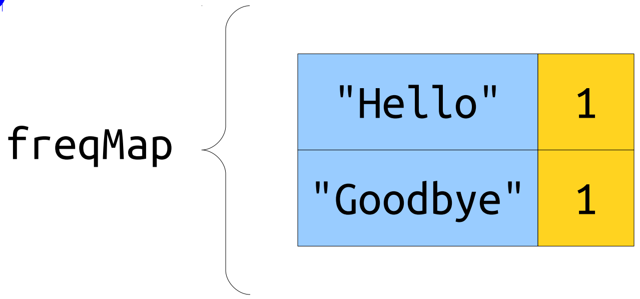
Map Autoinsertion

```
Map<string, int> freqMap;
while (true) {
    string text = getLine("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}
```



Map Autoinsertion

```
Map<string, int> freqMap;
while (true) {
    string text = getLine("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}
```



Sorting by First Letters

Map Autoinsertion

```
Lexicon english("EnglishWords.txt");

Map<char, Lexicon> wordsByFirstLetter;
for (string word: english) {
   wordsByFirstLetter[word[0]].add(word);
}
```

Map Autoinsertion

```
Lexicon english("EnglishWords.txt");

Map<char, Lexicon> wordsByFirstLetter;
for (string word: english) {
    wordsByFirstLetter[word[0]].add(word);
}
```

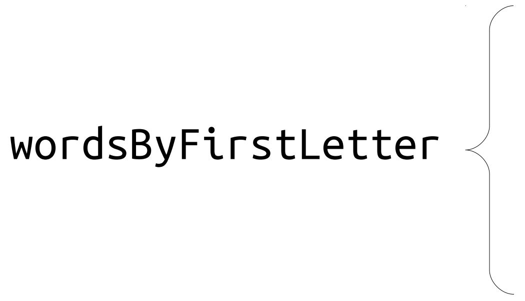
Map Autoinsertion

```
Lexicon english("EnglishWords.txt");

Map<char, Lexicon> wordsByFirstLetter;
for (string word: english) {
   wordsByFirstLetter[word[0]].add(word);
}
```

wordsByFirstLetter

Map Autoinsertion

```
Lexicon english("EnglishWords.txt");

Map<char, Lexicon> wordsByFirstLetter;
for (string word: english) {
    wordsByFirstLetter[word[0]].add(word);
}
```



Map Autoinsertion

```
Lexicon english("EnglishWords.txt");

Map<char, Lexicon> wordsByFirstLetter;
for (string word: english) {
    wordsByFirstLetter[word[0]].add(word);
}
```

wordsByFirstLetter
word
"first"

Map Autoinsertion

```
Lexicon english("EnglishWords.txt");

Map<char, Lexicon> wordsByFirstLetter;
for (string word: english) {
    wordsByFirstLetter[word[0]].add(word);
}
```

wordsByFirstLetter
word
"first"

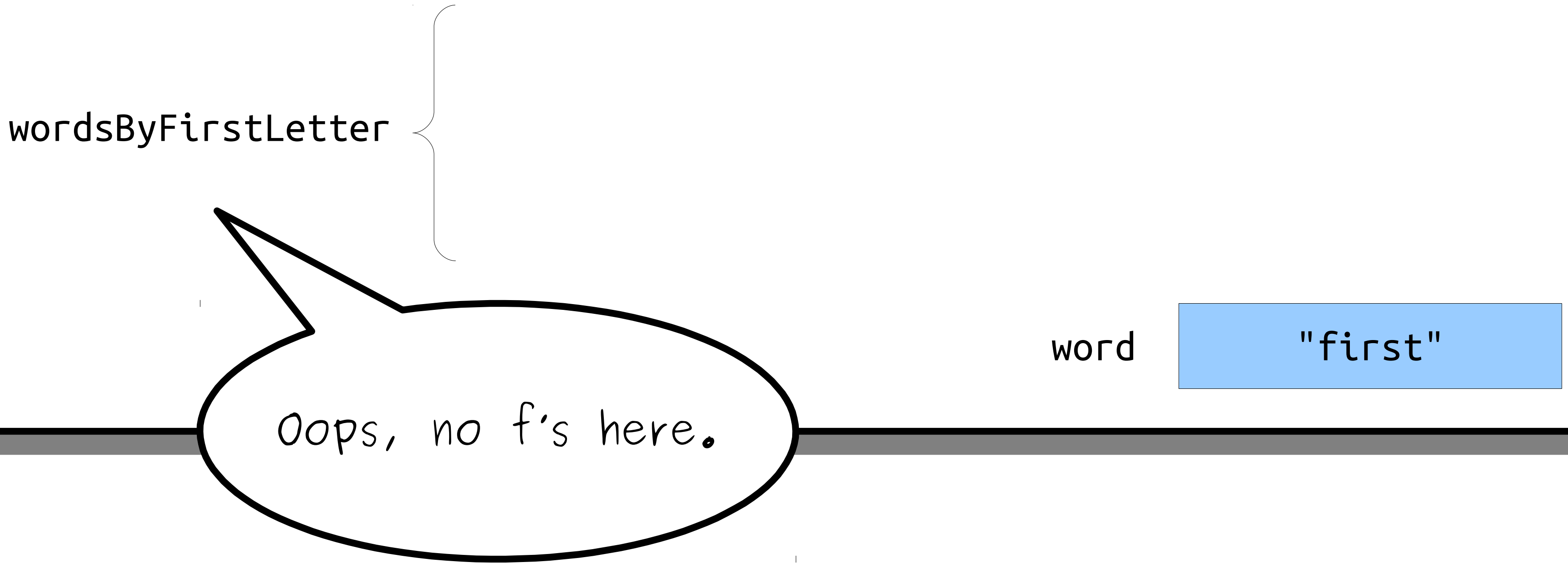
Map Autoinsertion

```
Lexicon english("EnglishWords.txt");

Map<char, Lexicon> wordsByFirstLetter;
for (string word: english) {
   wordsByFirstLetter[word[0]].add(word);
}
```

wordsByFirstLetter
word
"first"

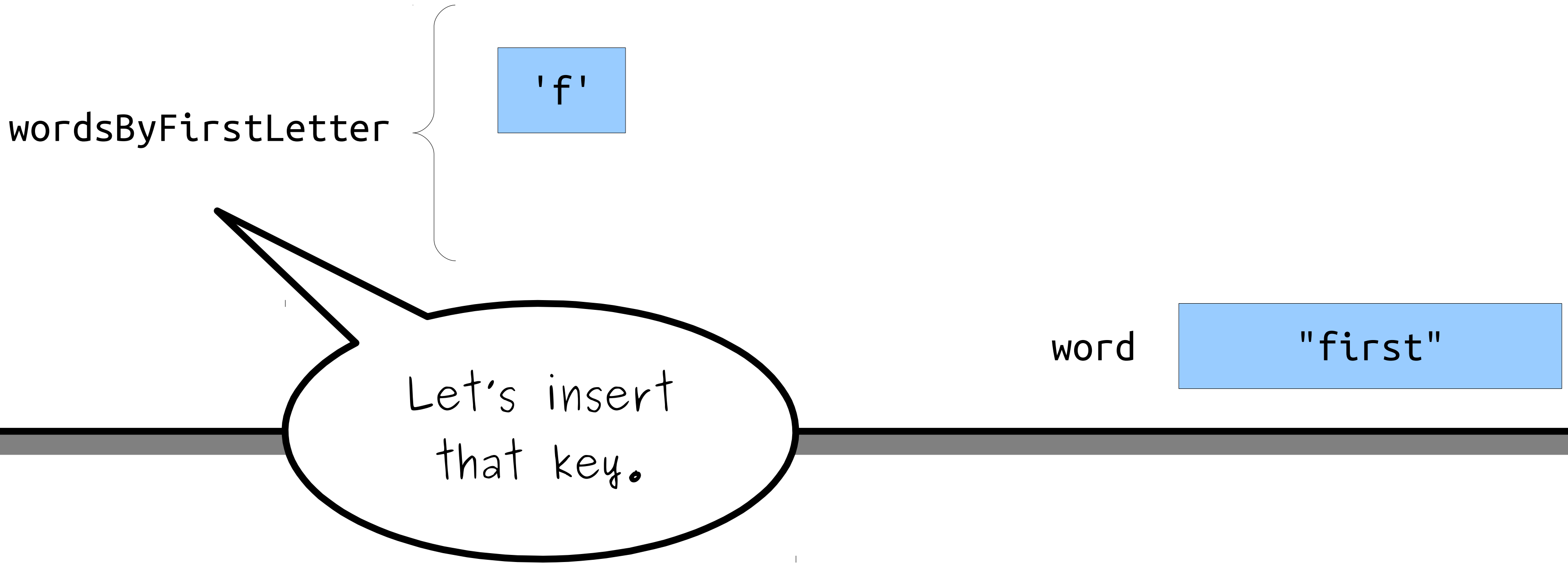
Map Autoinsertion

```
Lexicon english("EnglishWords.txt");

Map<char, Lexicon> wordsByFirstLetter;
for (string word: english) {
   wordsByFirstLetter[word[0]].add(word);
}
```



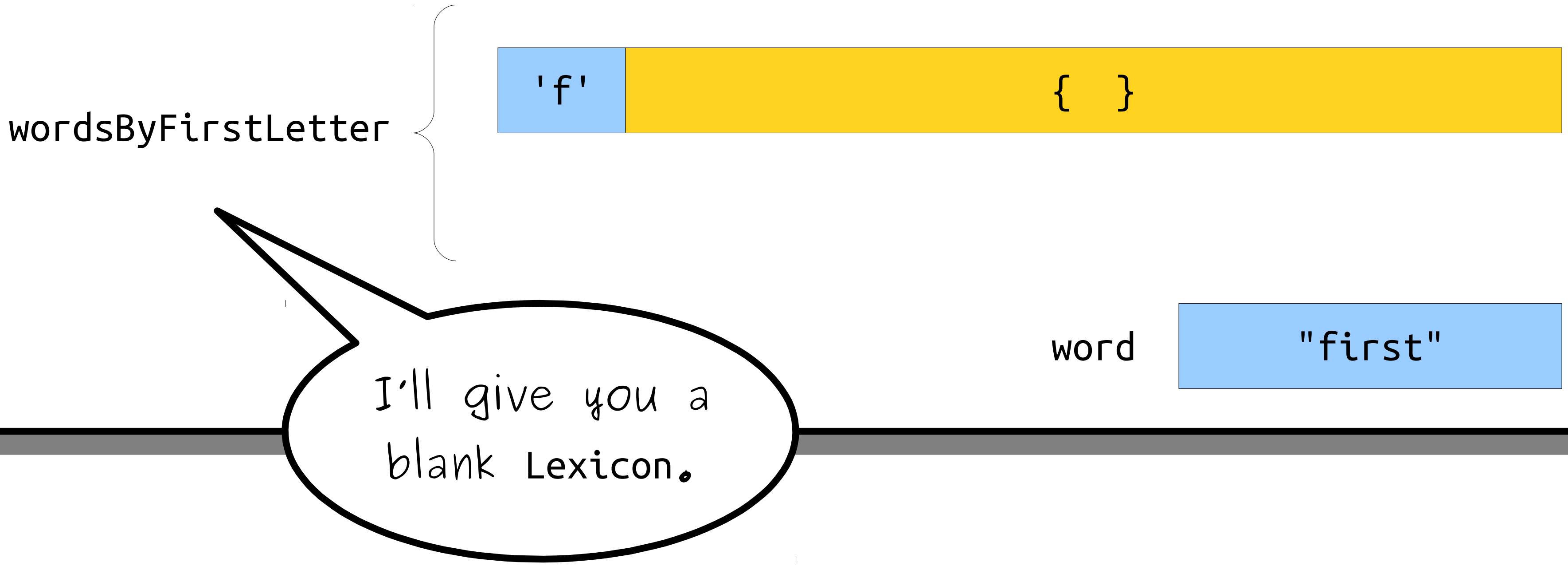
Map Autoinsertion

```
Lexicon english("EnglishWords.txt");

Map<char, Lexicon> wordsByFirstLetter;
for (string word: english) {
    wordsByFirstLetter[word[0]].add(word);
}
```



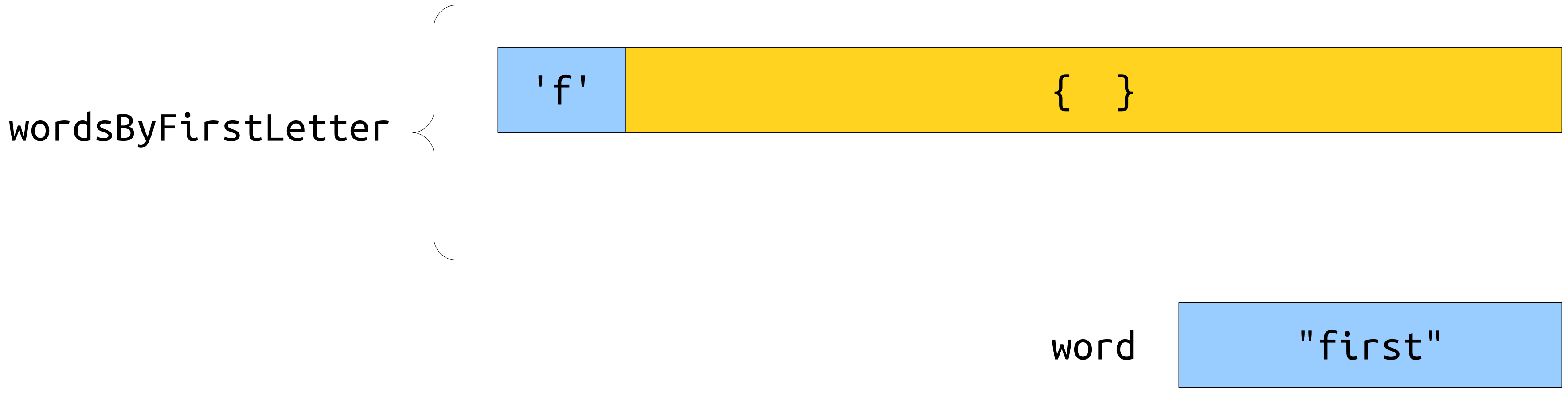
Map Autoinsertion

```
Lexicon english("EnglishWords.txt");

Map<char, Lexicon> wordsByFirstLetter;
for (string word: english) {
   wordsByFirstLetter[word[0]].add(word);
}
```



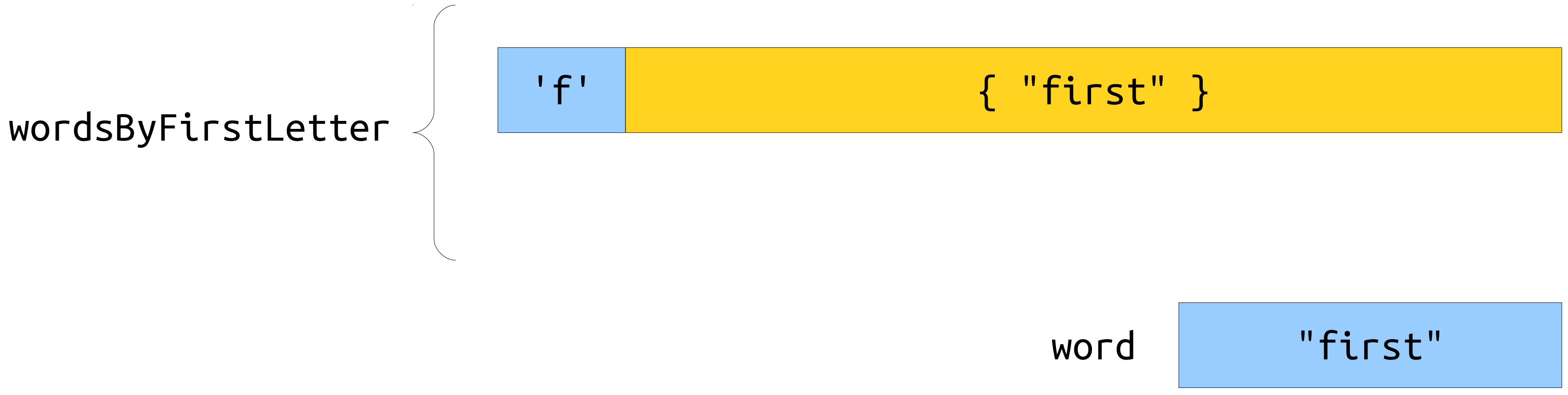
Map Autoinsertion

```
Lexicon english("EnglishWords.txt");

Map<char, Lexicon> wordsByFirstLetter;
for (string word: english) {
    wordsByFirstLetter[word[0]].add(word);
}
```



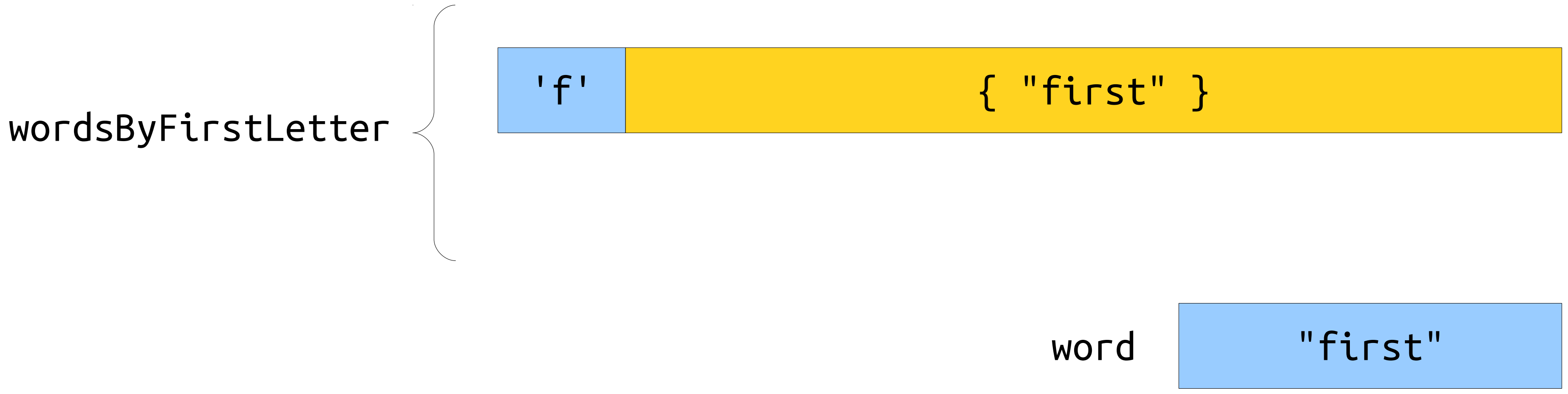
Map Autoinsertion

```
Lexicon english("EnglishWords.txt");

Map<char, Lexicon> wordsByFirstLetter;
for (string word: english) {
    wordsByFirstLetter[word[0]].add(word);
}
```



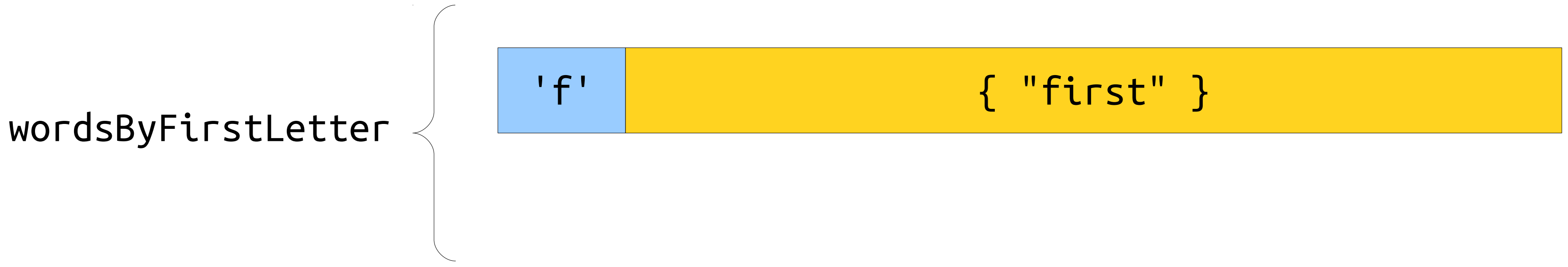
Map Autoinsertion

```
Lexicon english("EnglishWords.txt");

Map<char, Lexicon> wordsByFirstLetter;
for (string word: english) {
    wordsByFirstLetter[word[0]].add(word);
}
```



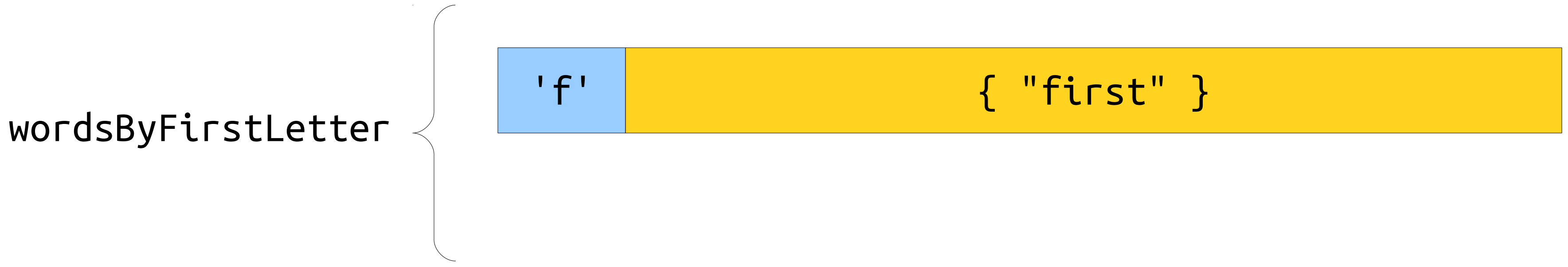
Map Autoinsertion

```
Lexicon english("EnglishWords.txt");

Map<char, Lexicon> wordsByFirstLetter;
for (string word: english) {
    wordsByFirstLetter[word[0]].add(word);
}
```



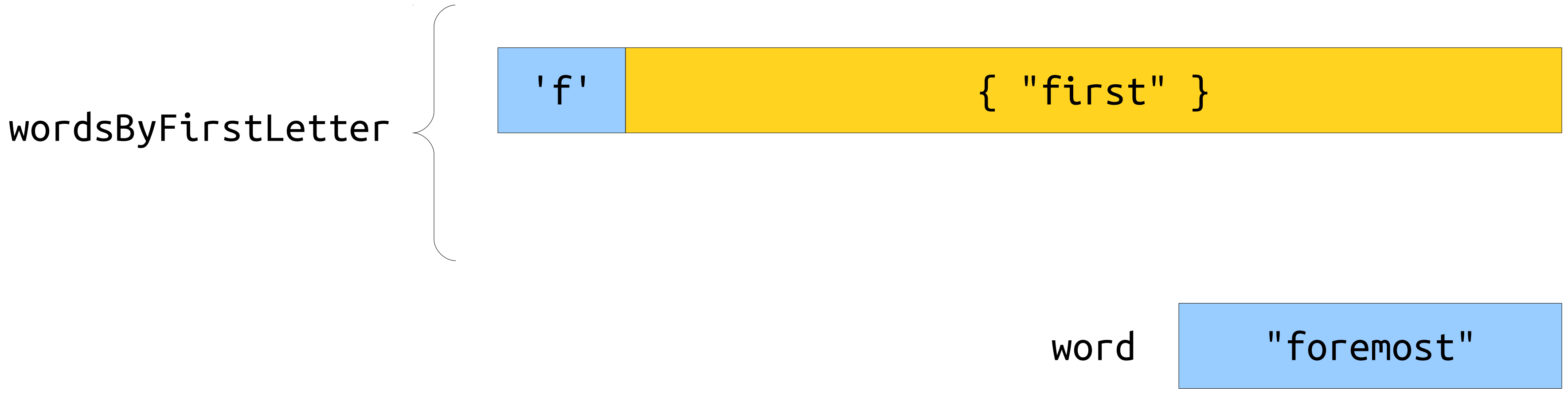
Map Autoinsertion

```
Lexicon english("EnglishWords.txt");

Map<char, Lexicon> wordsByFirstLetter;
for (string word: english) {
    wordsByFirstLetter[word[0]].add(word);
}
```



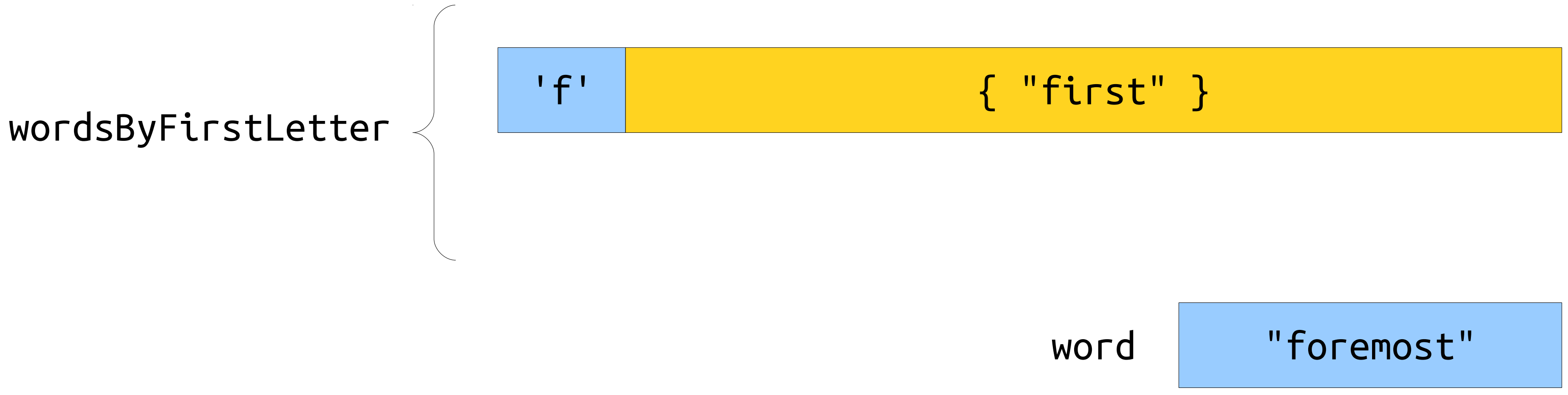
Map Autoinsertion

```
Lexicon english("EnglishWords.txt");

Map<char, Lexicon> wordsByFirstLetter;
for (string word: english) {
    wordsByFirstLetter[word[0]].add(word);
}
```



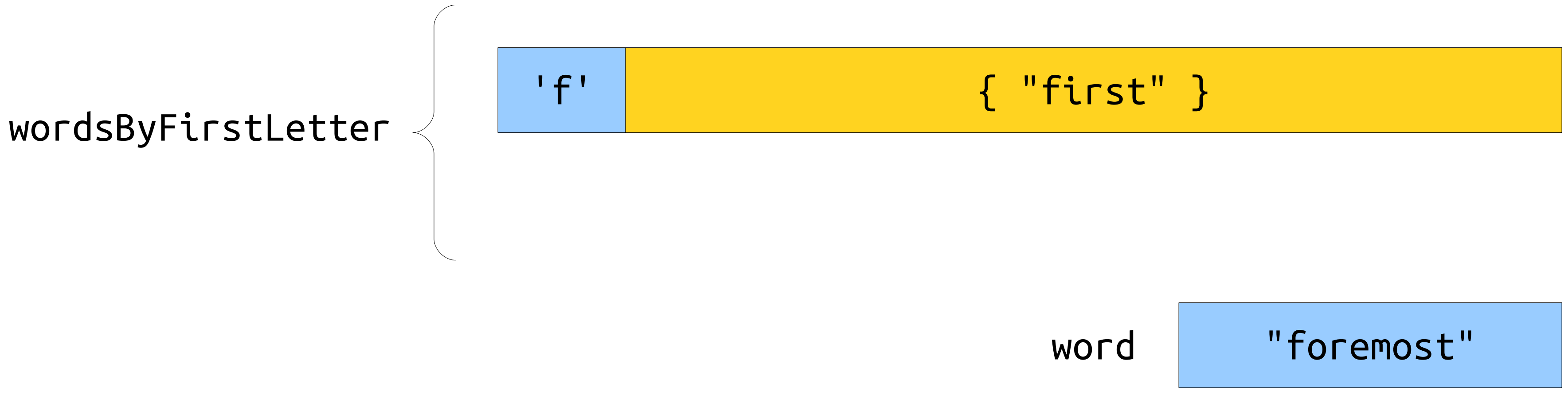
Map Autoinsertion

```
Lexicon english("EnglishWords.txt");

Map<char, Lexicon> wordsByFirstLetter;
for (string word: english) {
    wordsByFirstLetter[word[0]].add(word);
}
```



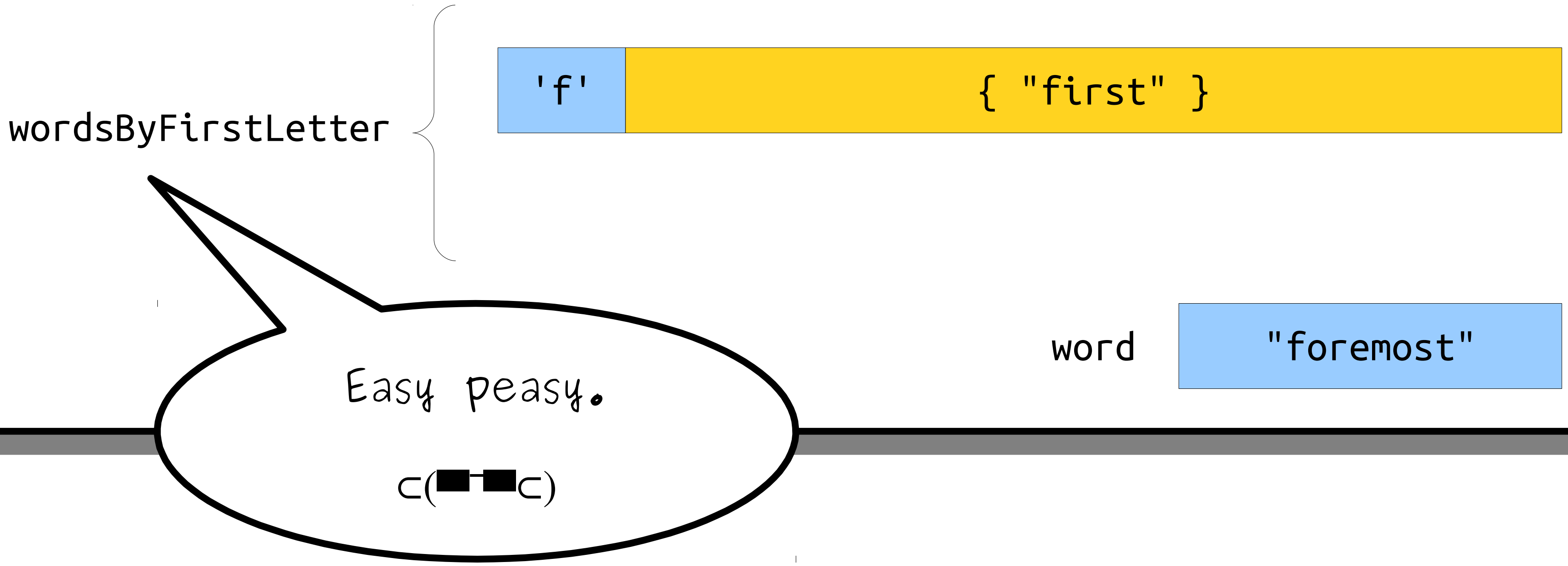
Map Autoinsertion

```
Lexicon english("EnglishWords.txt");

Map<char, Lexicon> wordsByFirstLetter;
for (string word: english) {
    wordsByFirstLetter[word[0]].add(word);
}
```



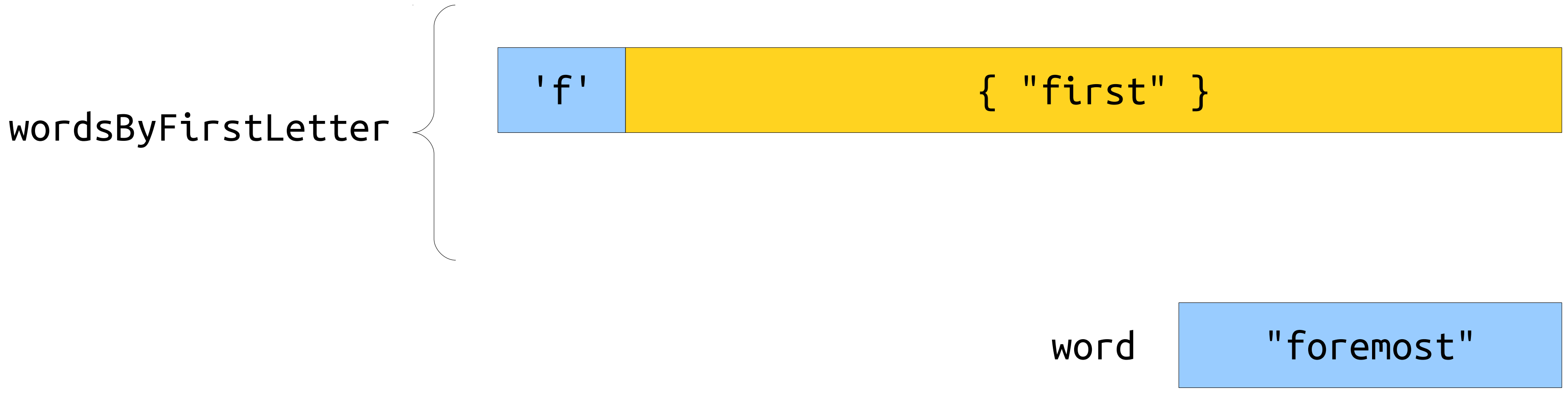
Map Autoinsertion

```
Lexicon english("EnglishWords.txt");

Map<char, Lexicon> wordsByFirstLetter;
for (string word: english) {
    wordsByFirstLetter[word[0]].add(word);
}
```



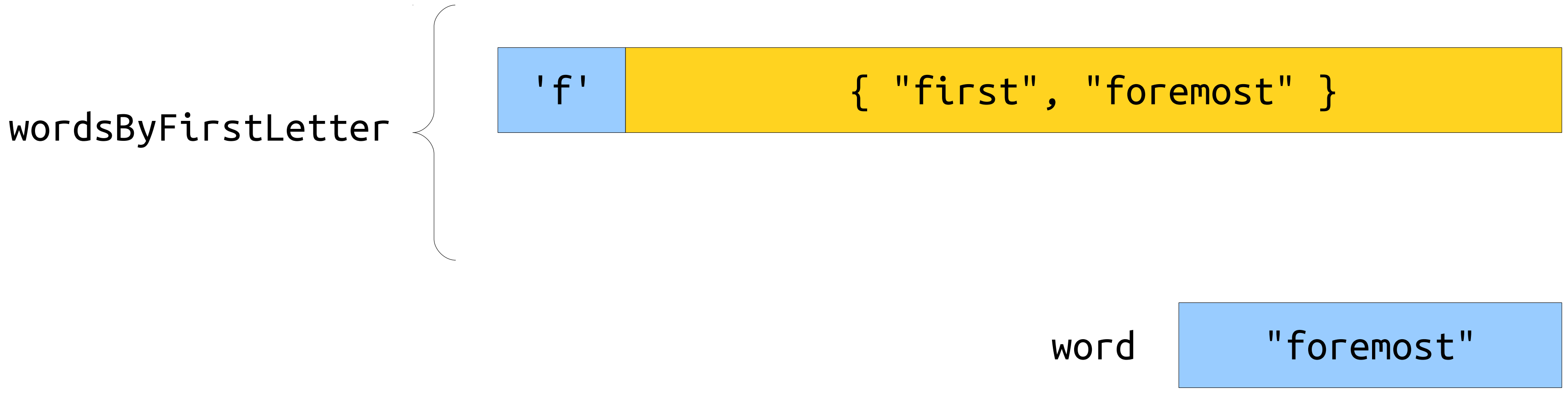
Map Autoinsertion

```
Lexicon english("EnglishWords.txt");

Map<char, Lexicon> wordsByFirstLetter;
for (string word: english) {
    wordsByFirstLetter[word[0]].add(word);
}
```



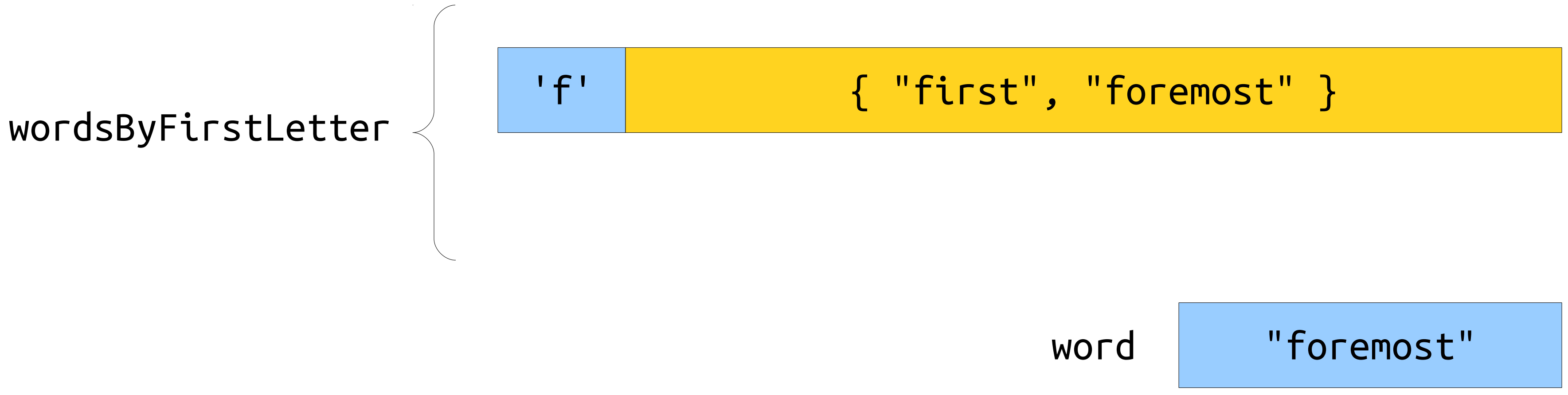
Map Autoinsertion

```
Lexicon english("EnglishWords.txt");

Map<char, Lexicon> wordsByFirstLetter;
for (string word: english) {
   wordsByFirstLetter[word[0]].add(word);
}
```



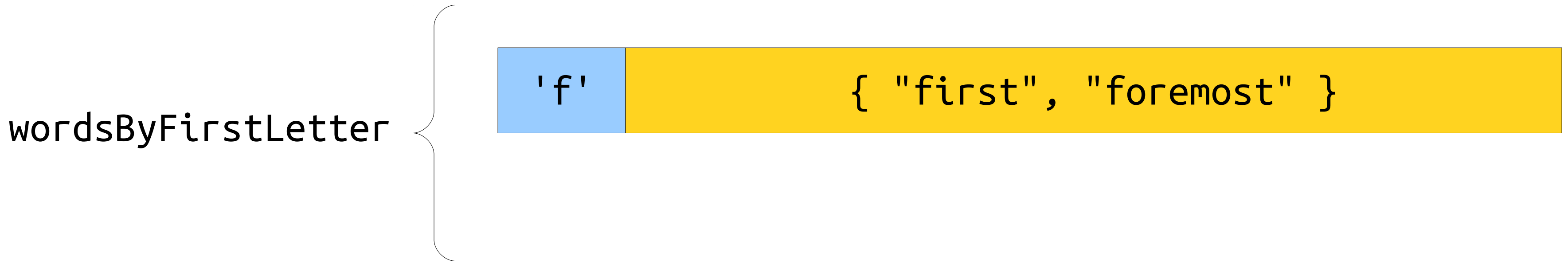
Map Autoinsertion

```
Lexicon english("EnglishWords.txt");

Map<char, Lexicon> wordsByFirstLetter;
for (string word: english) {
    wordsByFirstLetter[word[0]].add(word);
}
```



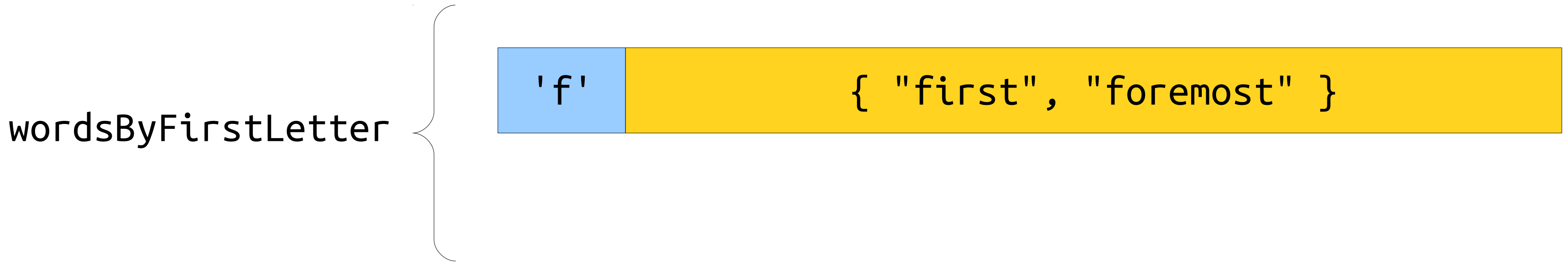
Map Autoinsertion

```
Lexicon english("EnglishWords.txt");

Map<char, Lexicon> wordsByFirstLetter;
for (string word: english) {
    wordsByFirstLetter[word[0]].add(word);
}
```



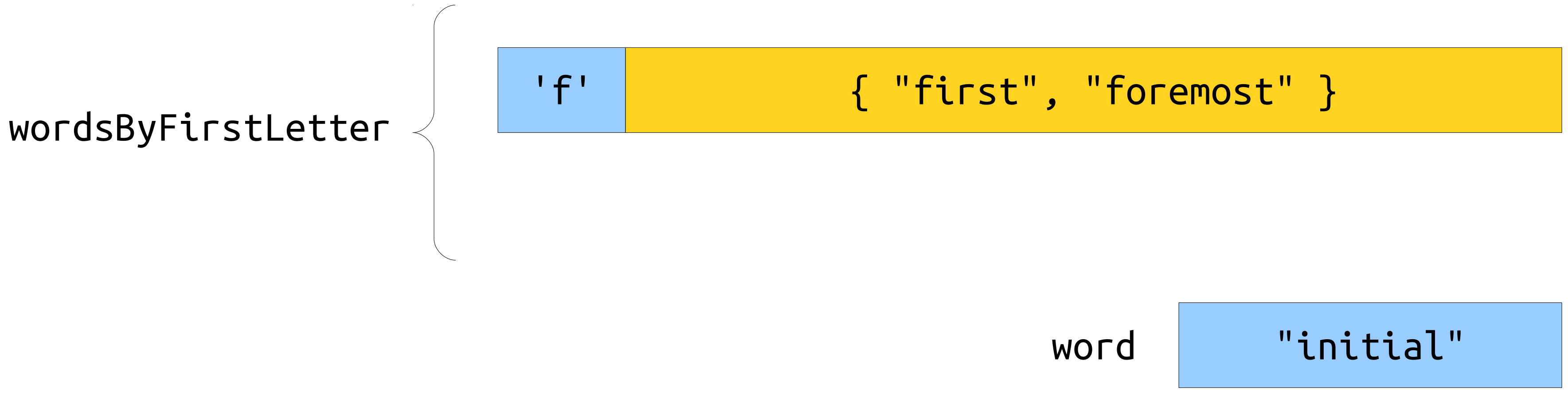
Map Autoinsertion

```
Lexicon english("EnglishWords.txt");

Map<char, Lexicon> wordsByFirstLetter;
for (string word: english) {
    wordsByFirstLetter[word[0]].add(word);
}
```



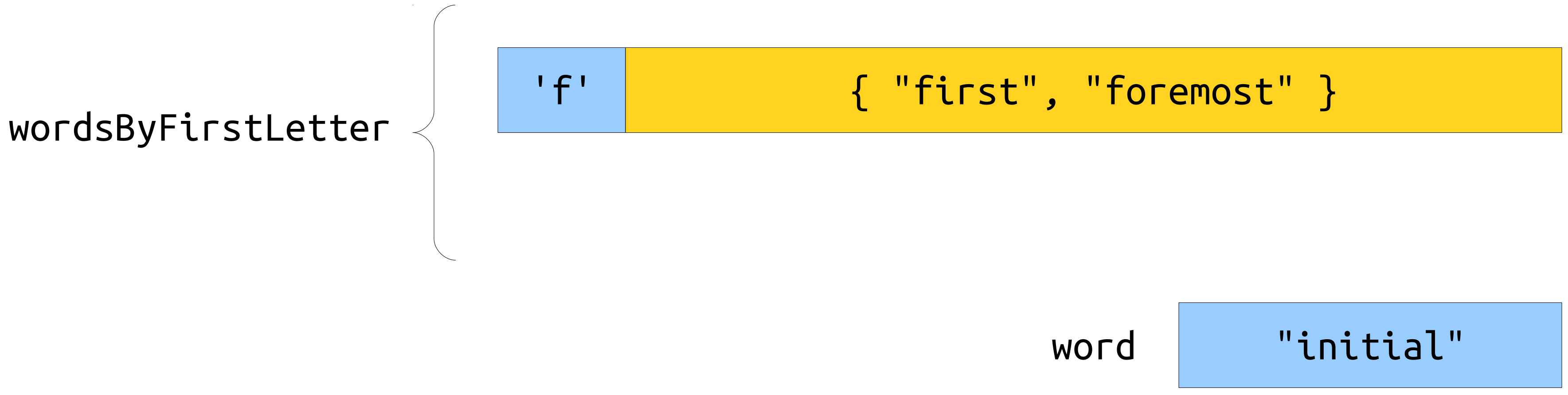
Map Autoinsertion

```
Lexicon english("EnglishWords.txt");

Map<char, Lexicon> wordsByFirstLetter;
for (string word: english) {
    wordsByFirstLetter[word[0]].add(word);
}
```



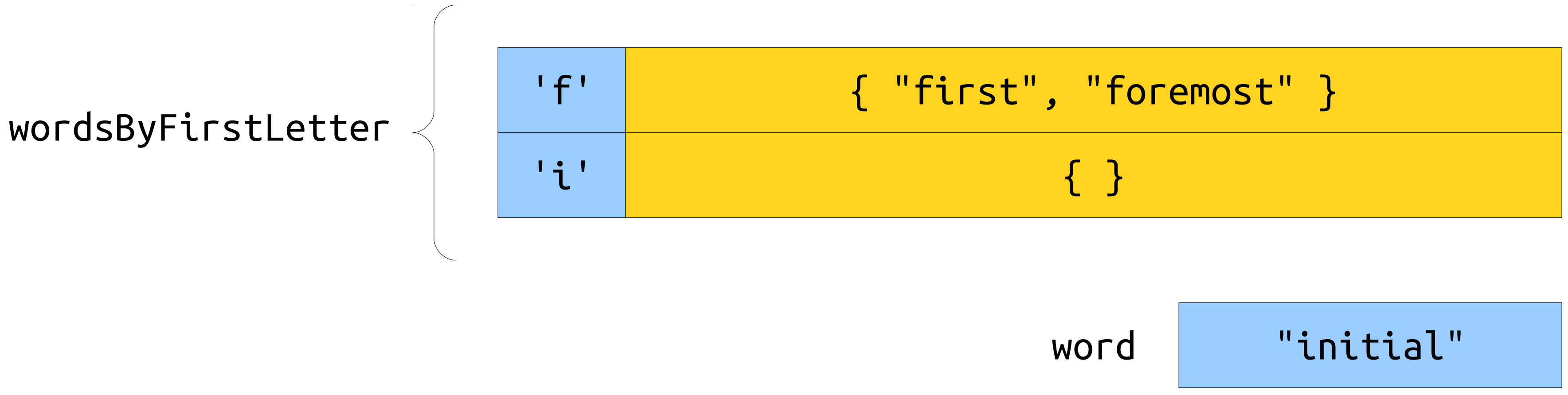
Map Autoinsertion

```
Lexicon english("EnglishWords.txt");

Map<char, Lexicon> wordsByFirstLetter;
for (string word: english) {
    wordsByFirstLetter[word[0]].add(word);
}
```



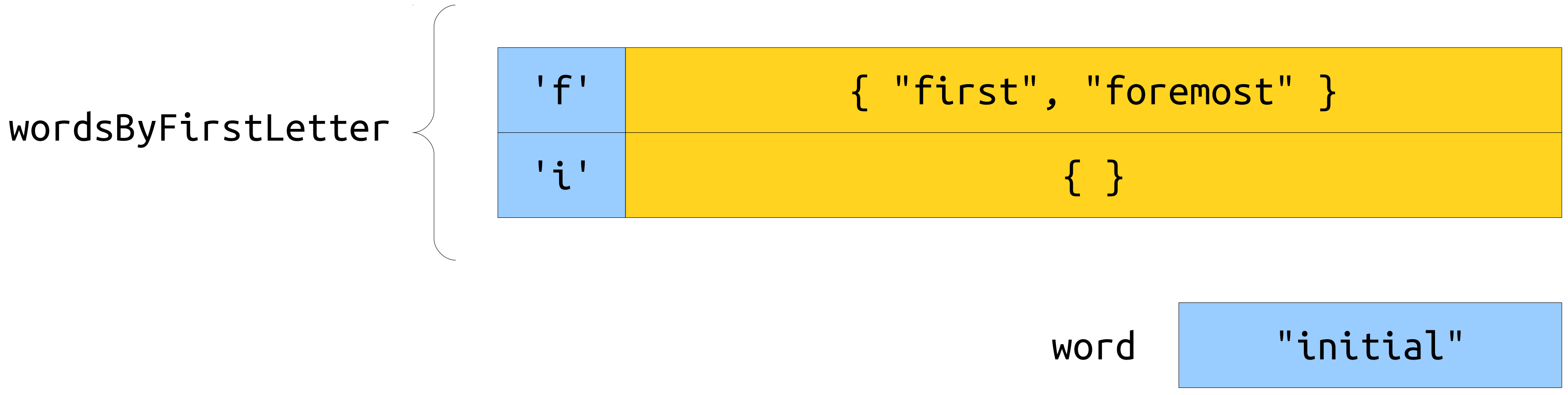
Map Autoinsertion

```
Lexicon english("EnglishWords.txt");

Map<char, Lexicon> wordsByFirstLetter;
for (string word: english) {
    wordsByFirstLetter[word[0]].add(word);
}
```



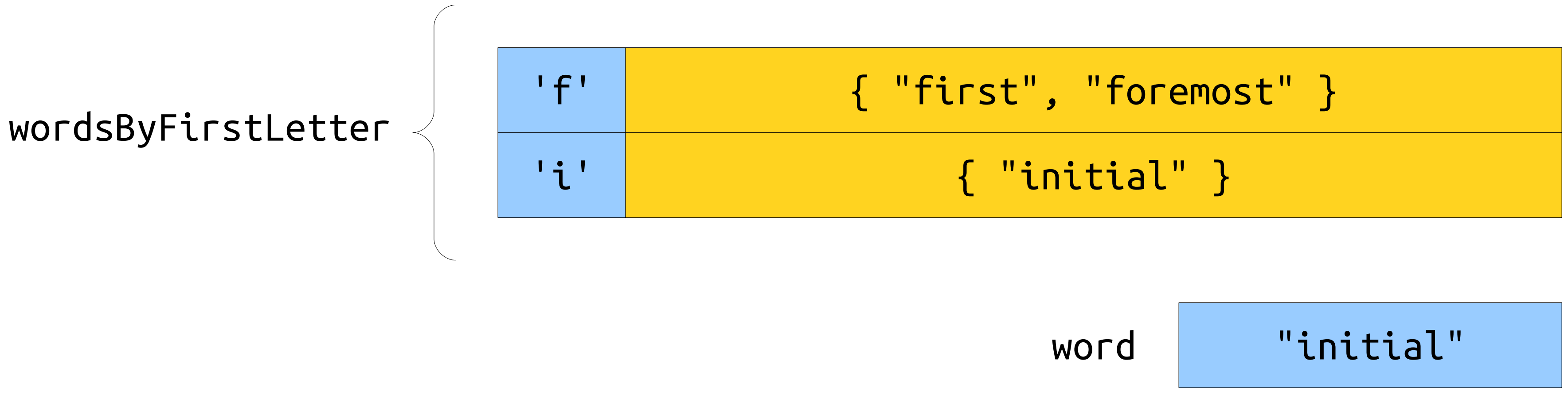
Map Autoinsertion

```
Lexicon english("EnglishWords.txt");

Map<char, Lexicon> wordsByFirstLetter;
for (string word: english) {
    wordsByFirstLetter[word[0]].add(word);
}
```



Map Autoinsertion

```
Lexicon english("EnglishWords.txt");

Map<char, Lexicon> wordsByFirstLetter;
for (string word: english) {
    wordsByFirstLetter[word[0]].add(word);
}
```



Anagrams

- Two words are ***anagrams*** of one another if the letters in one can be rearranged into the other.
- Some examples:
  - “Senator” and “treason.”
  - “Praising” and “aspiring.”
  - “Arrogant” and “tarragon.”
- ***Question for you:*** does this concept exist in other languages? If so, please send me examples!

Anagrams

- ***Nifty fact:*** two words are anagrams if you get the same string when you write the letters in those words in sorted order.
- For example, “praising” and “aspiring” are anagrams because, in both cases, you get the string “aiignprs” if you sort the letters.

Anagram Clusters

- Let’s group all words in English into “clusters” of words that are all anagrams of one another.
- We’ll use a `Map<string, Lexicon>`.
  - Each key is a string of letters in sorted order.
  - Each value is the collection of English words that have those letters in that order.

Next Time

- ***Thinking Recursively***
  - How can you best solve problems using recursion?
  - What techniques are necessary to do so?
  - And what problems yield easily to a recursive solution?

Extra Content: How to Sort a String

Order in Range-Based `for` Loops

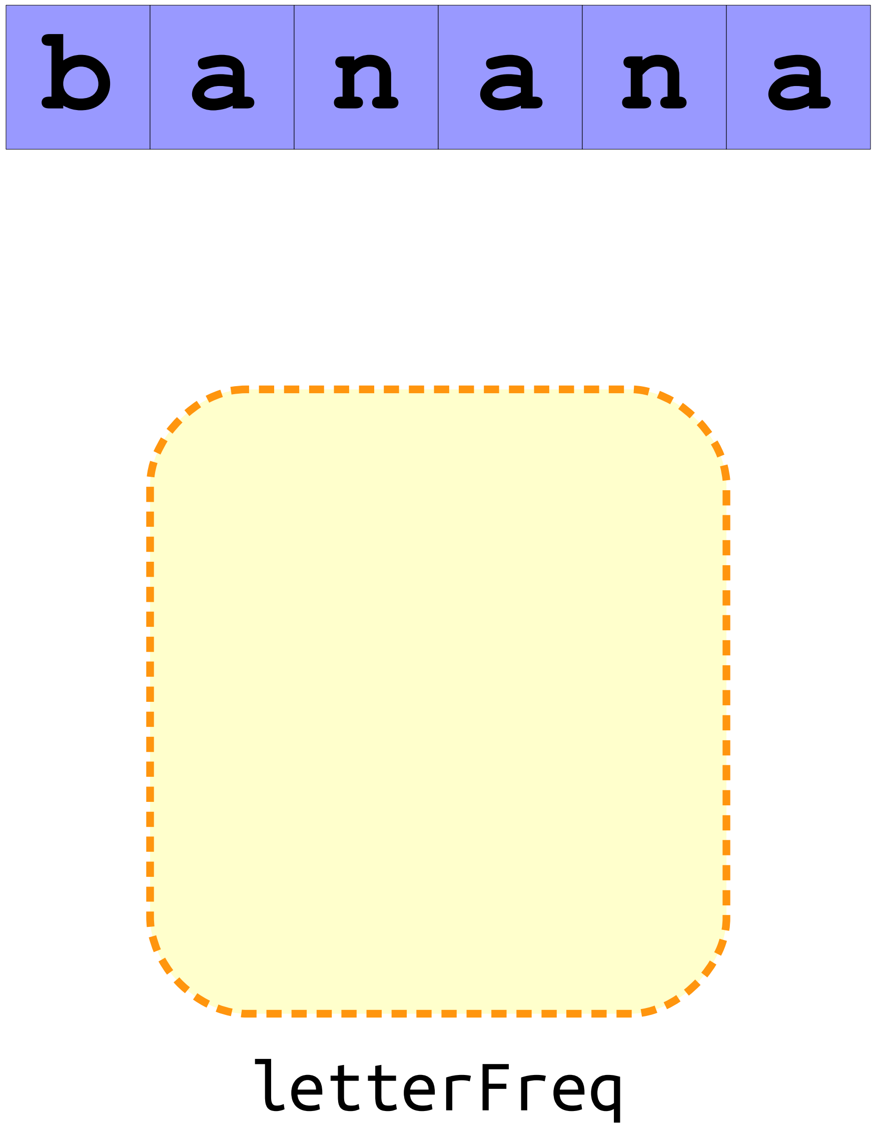
- When using the range-based for loop to iterate over a collection:
    - In a `Vector`, `string`, or array, the elements are retrieved in order.
    - In a `Map`, the *keys* are returned in sorted order.
    - In a `Set` or `Lexicon`, the values are returned in sorted order.

Counting Sort

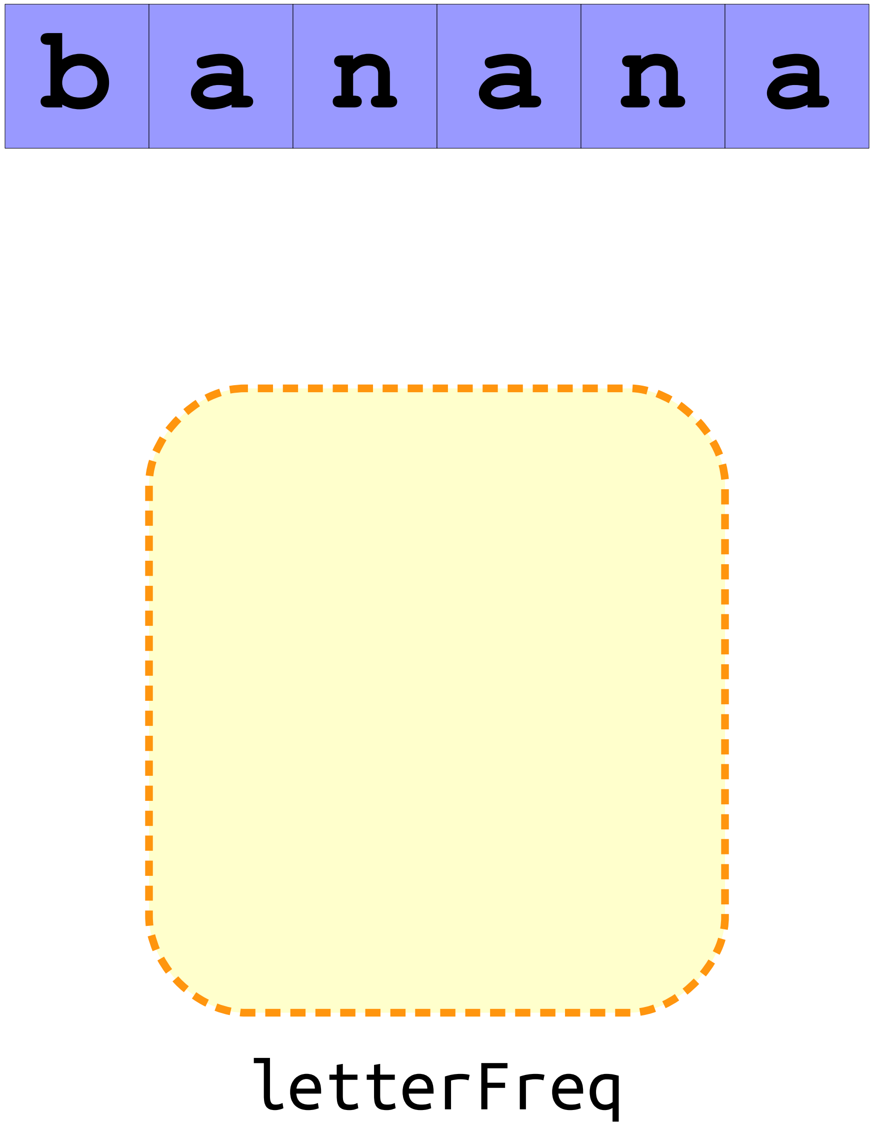
Counting Sort

b	a	n	a	n	a

Counting Sort

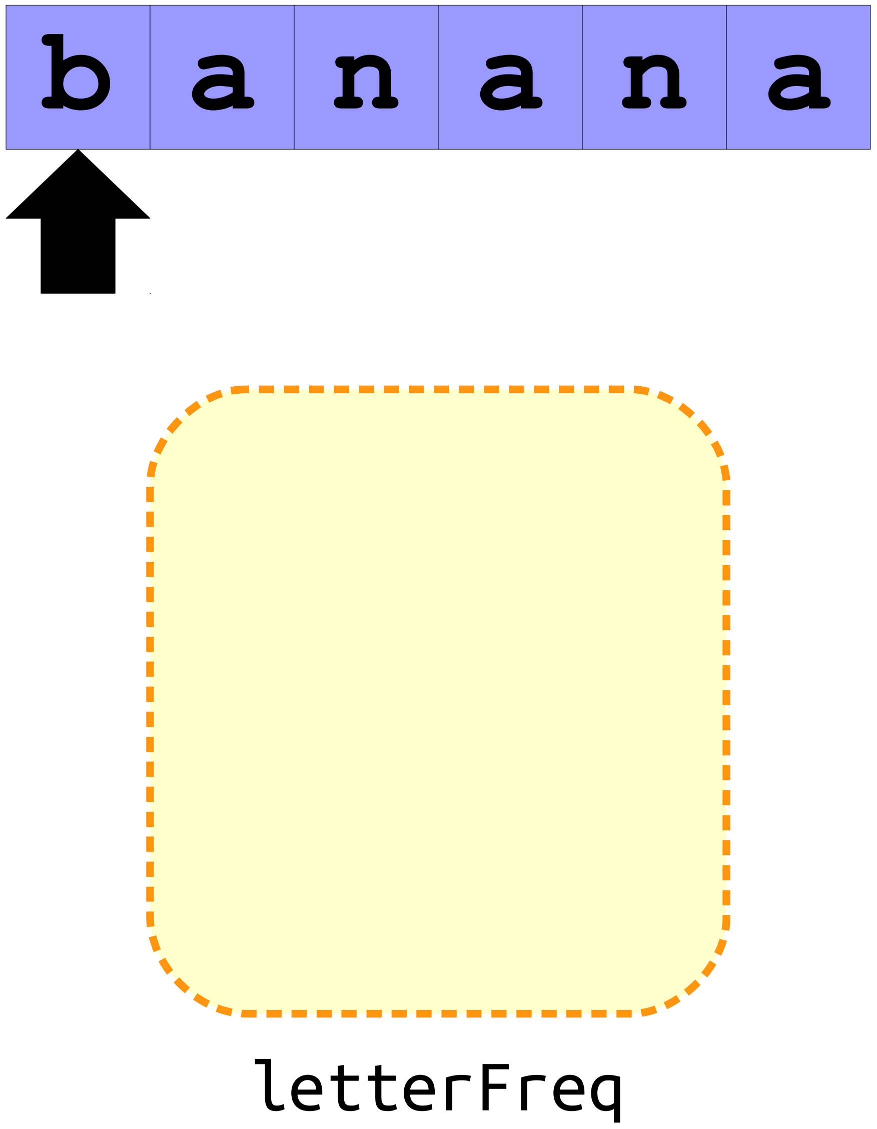


Counting Sort



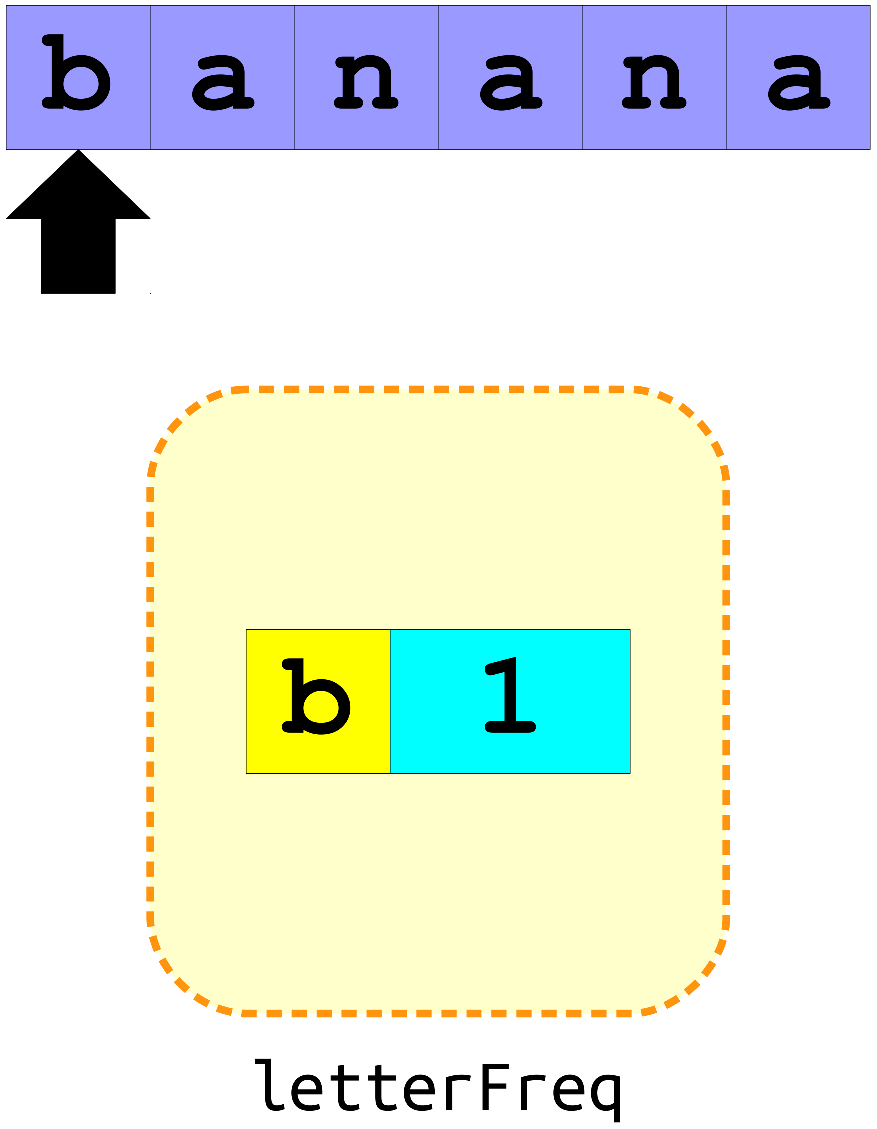
```
for (char ch: input) {
    letterFreq[ch]++;
}
```

Counting Sort



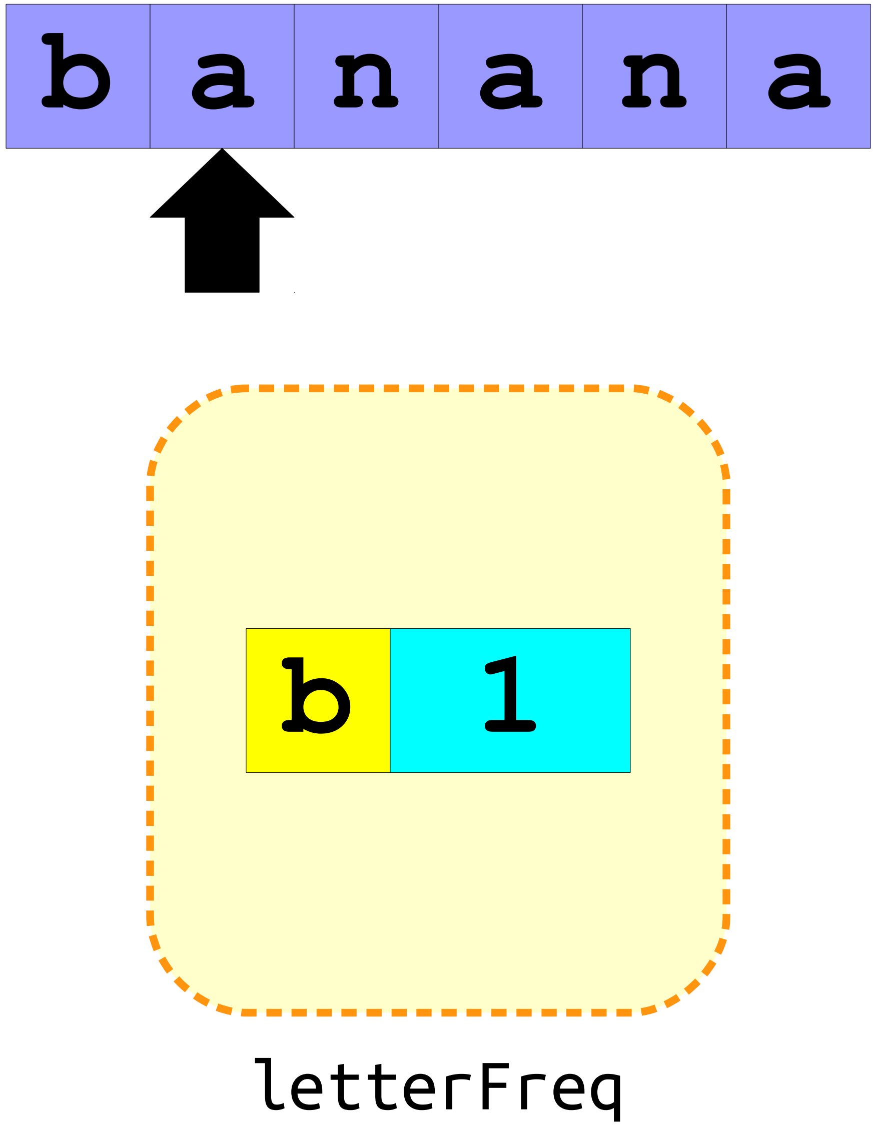
```
for (char ch: input) {
    letterFreq[ch]++;
}
```

Counting Sort



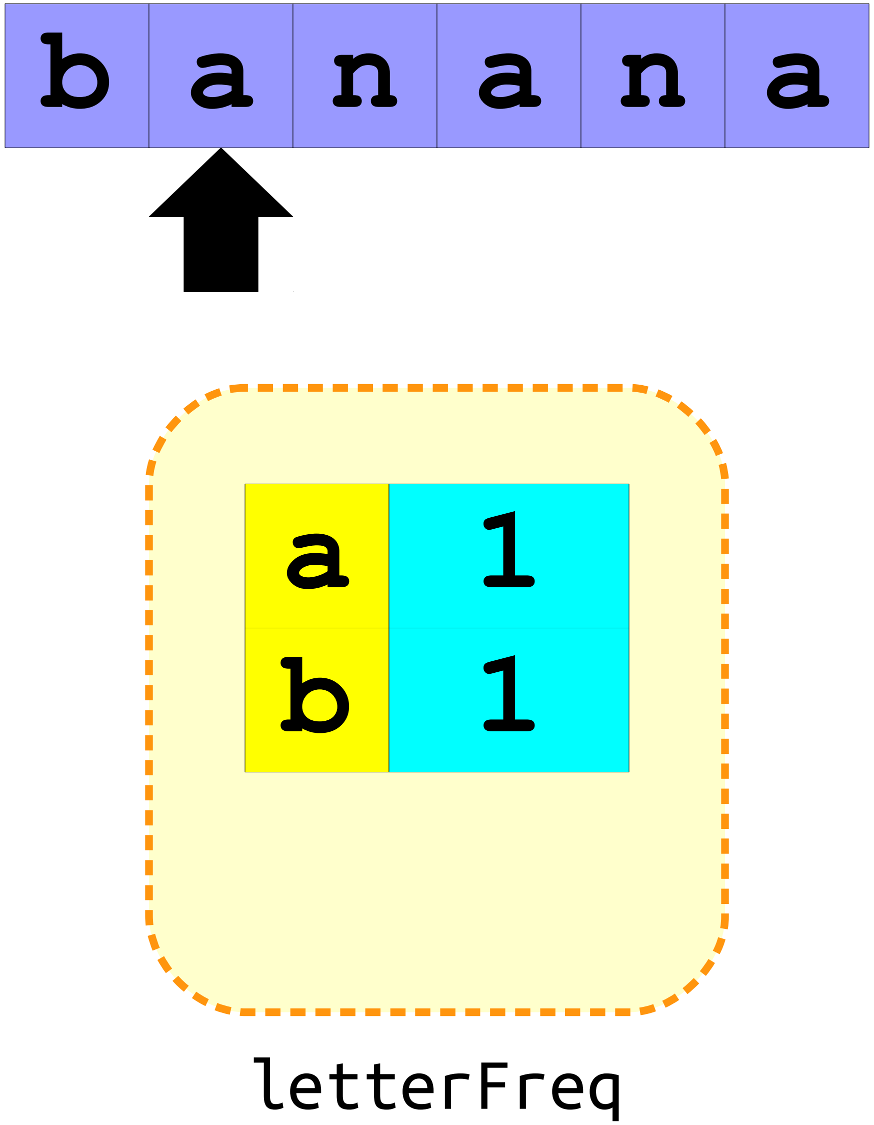
```
for (char ch: input) {
    letterFreq[ch]++;
}
```

Counting Sort



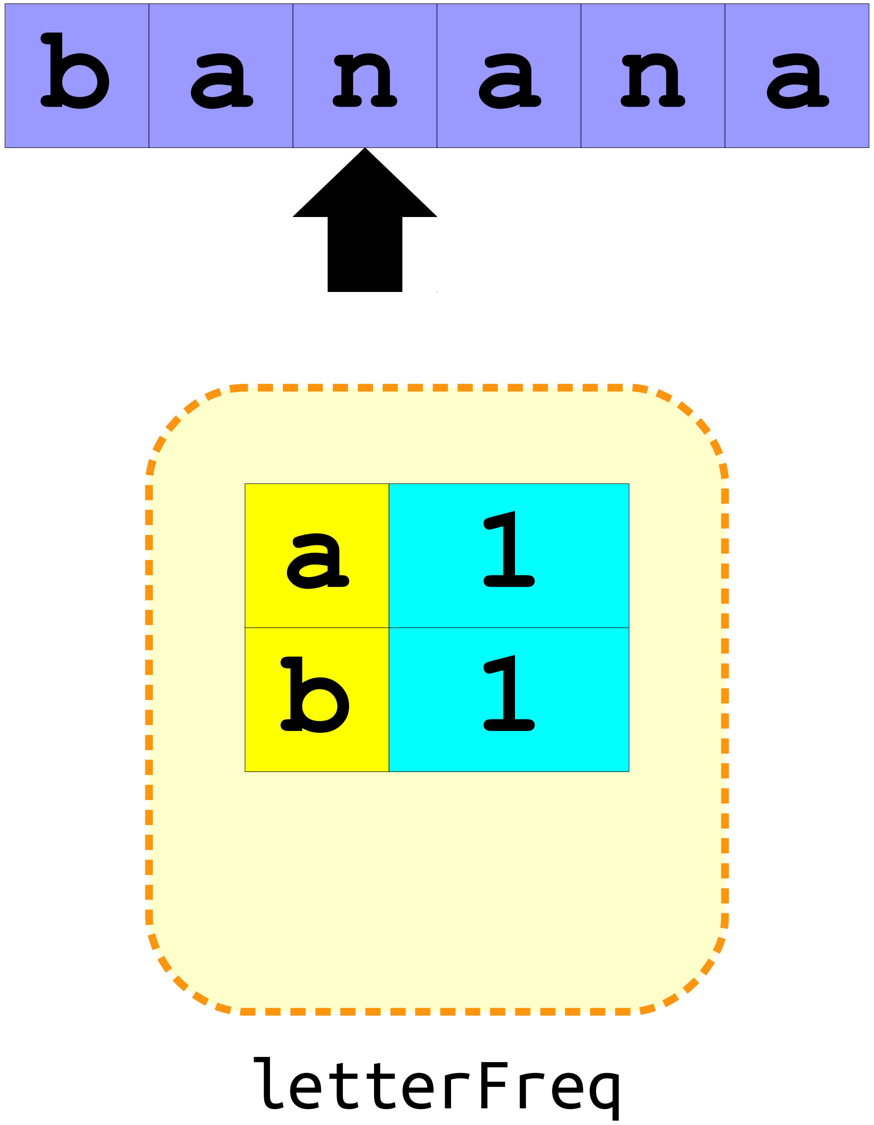
```
for (char ch: input) {
    letterFreq[ch]++;
}
```

Counting Sort



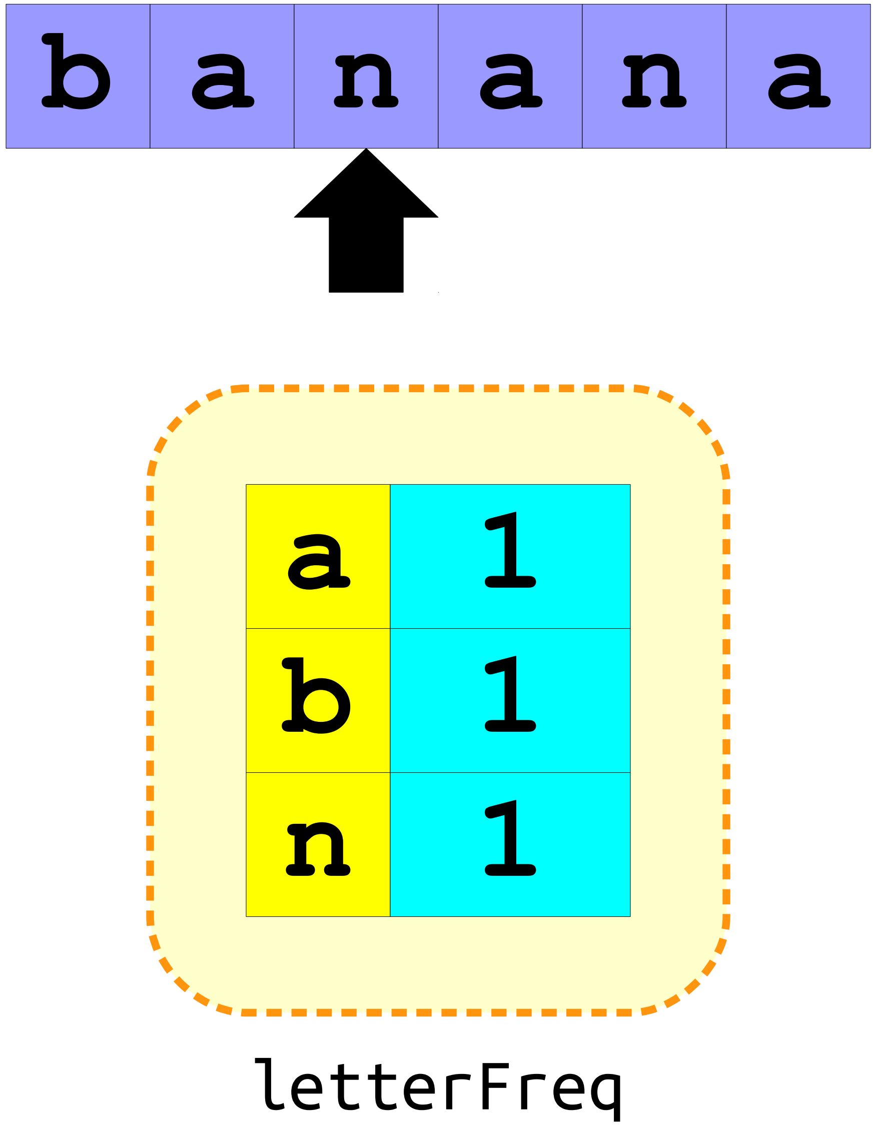
```
for (char ch: input) {
    letterFreq[ch]++;
}
```

Counting Sort



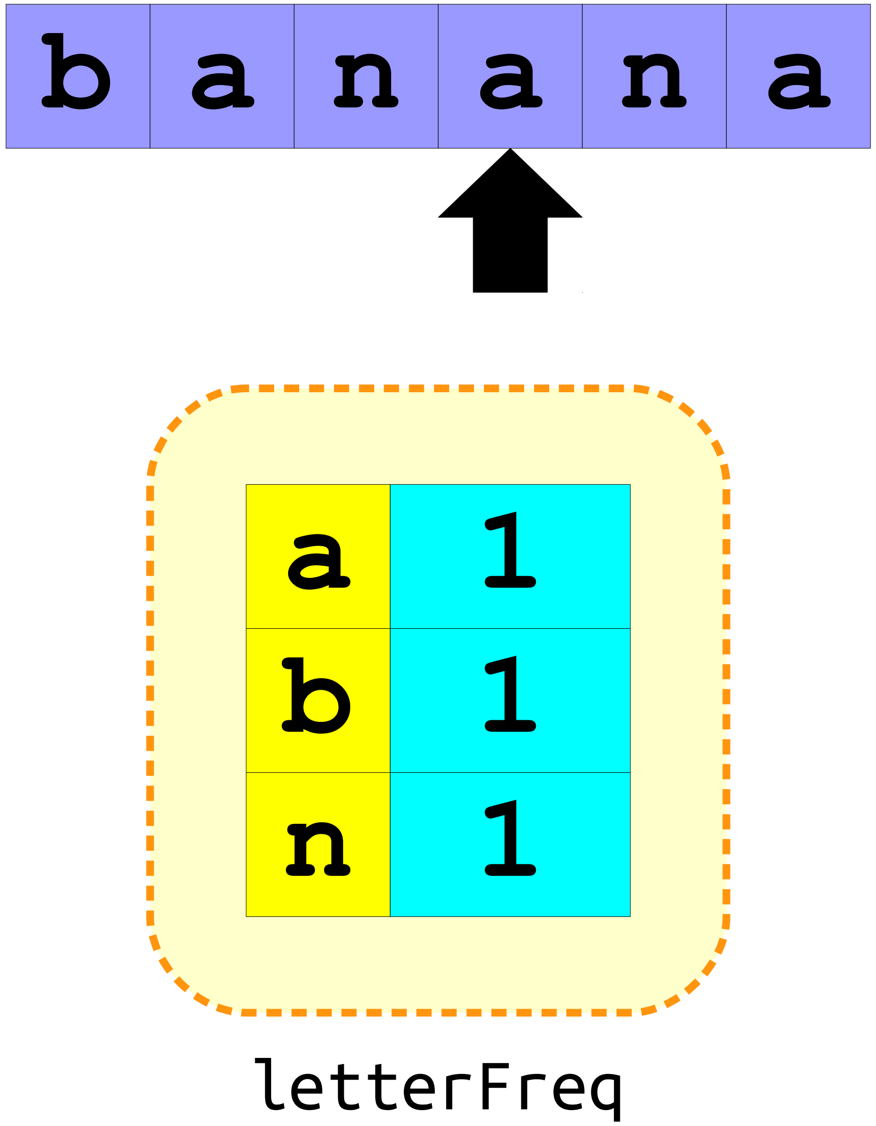
```
for (char ch: input) {
    letterFreq[ch]++;
}
```

Counting Sort



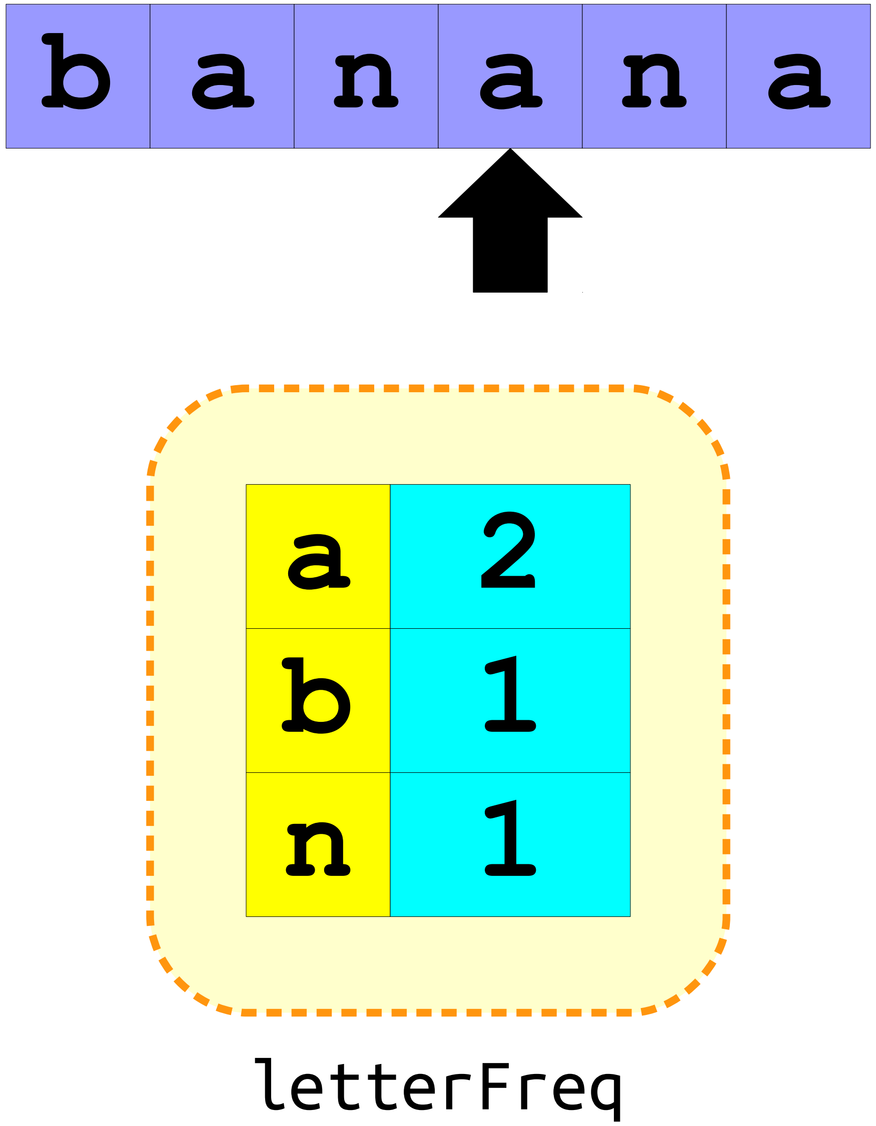
```
for (char ch: input) {
    letterFreq[ch]++;
}
```

Counting Sort



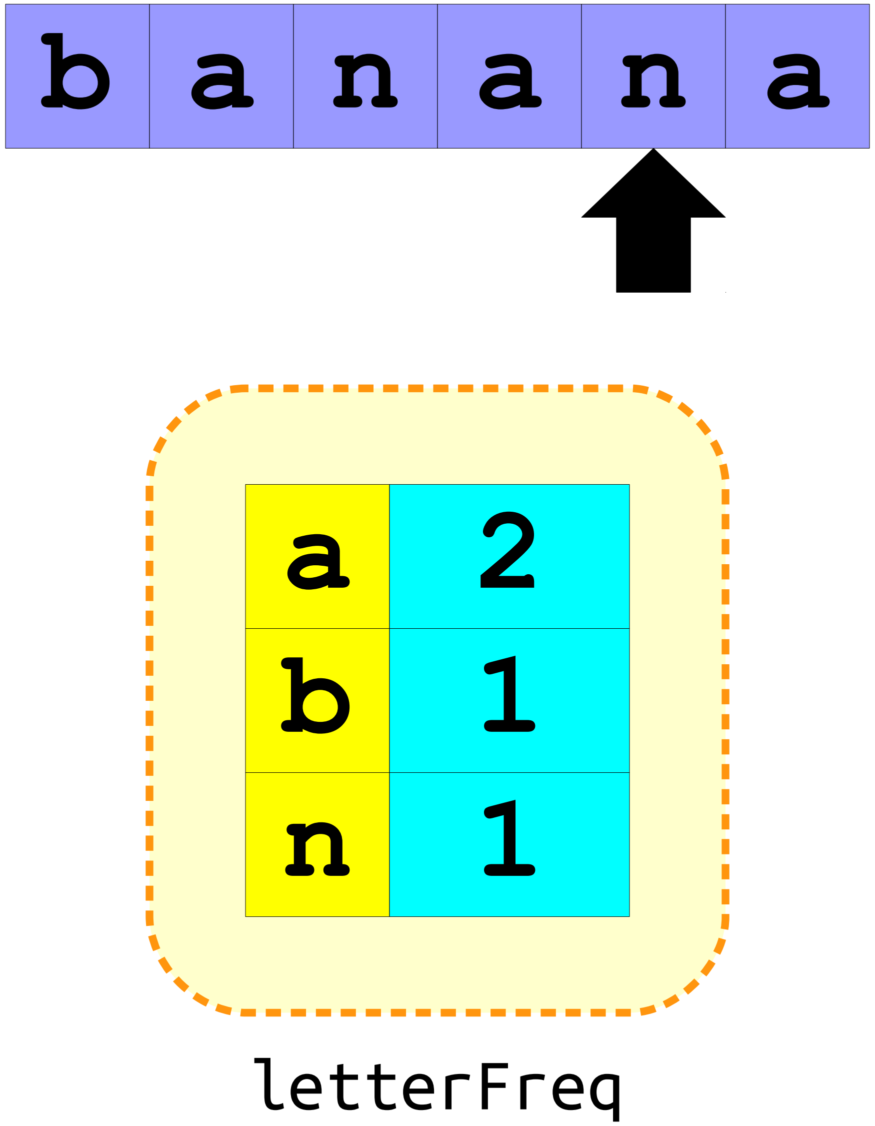
```
for (char ch: input) {
    letterFreq[ch]++;
}
```

Counting Sort



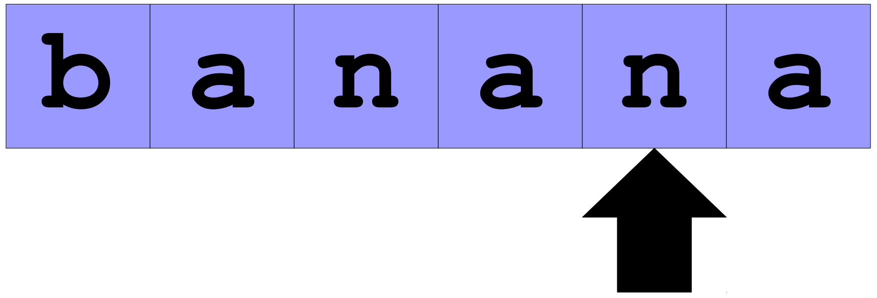
```
for (char ch: input) {
    letterFreq[ch]++;
}
```

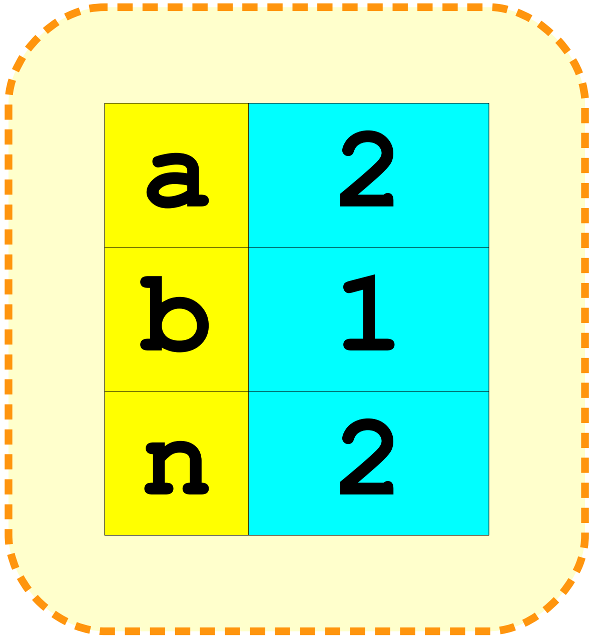
Counting Sort



```
for (char ch: input) {
    letterFreq[ch]++;
}
```

Counting Sort



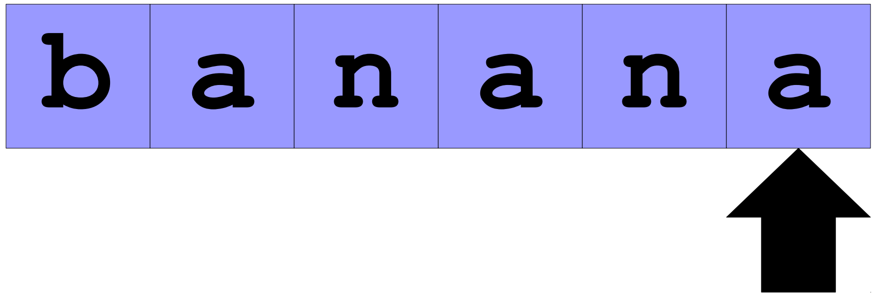


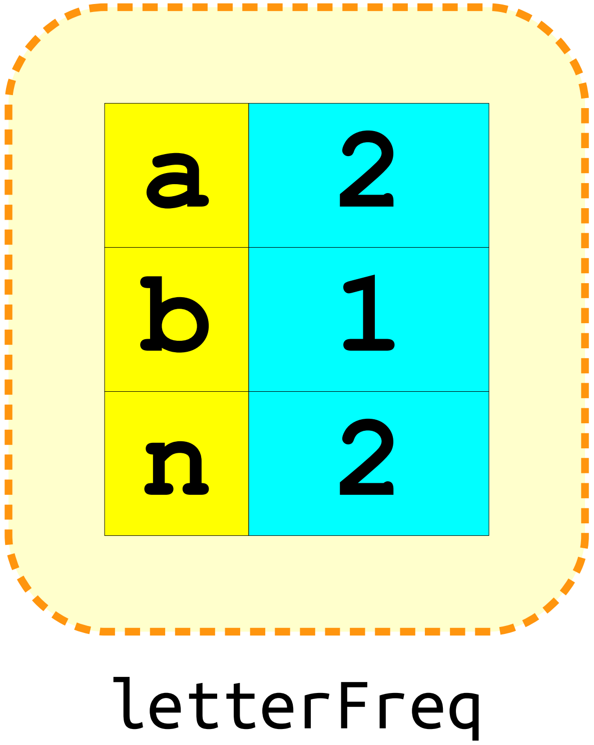
a	2
b	1
n	2

letterFreq

```
for (char ch: input) {
    letterFreq[ch]++;
}
```

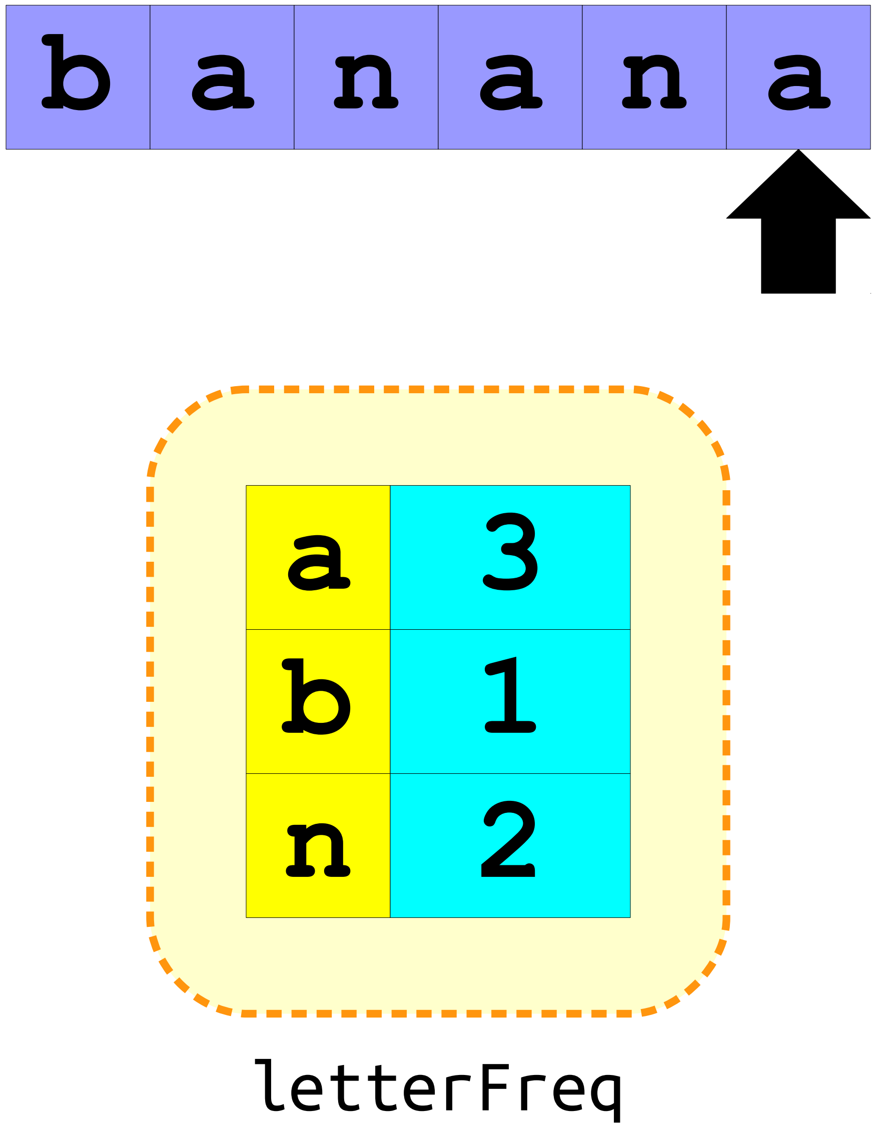
Counting Sort





```
for (char ch: input) {
    letterFreq[ch]++;
}
```

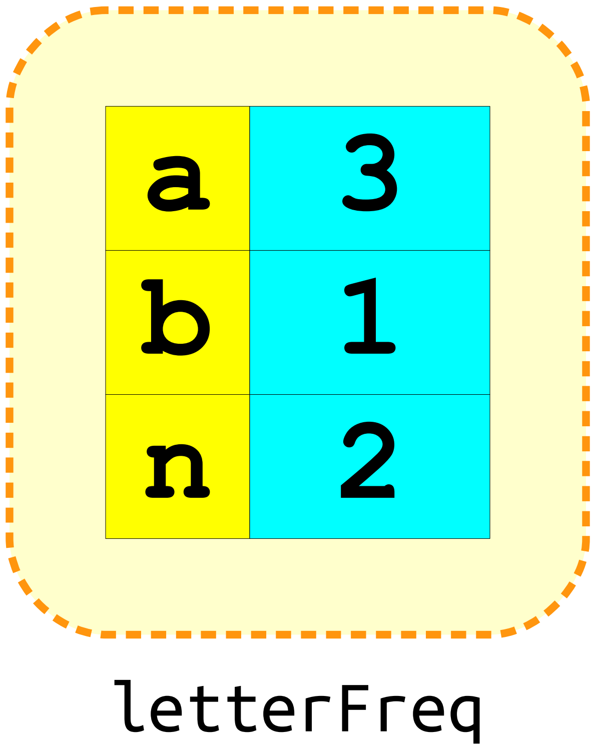
Counting Sort



```
for (char ch: input) {
    letterFreq[ch]++;
}
```

Counting Sort

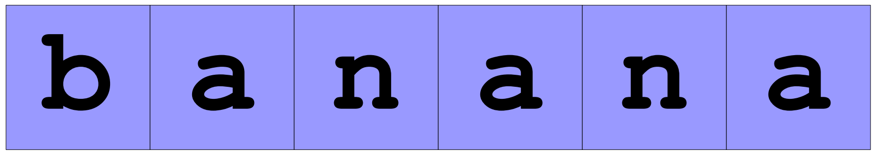
b	**a**	**n**	**a**	**n**	**a**

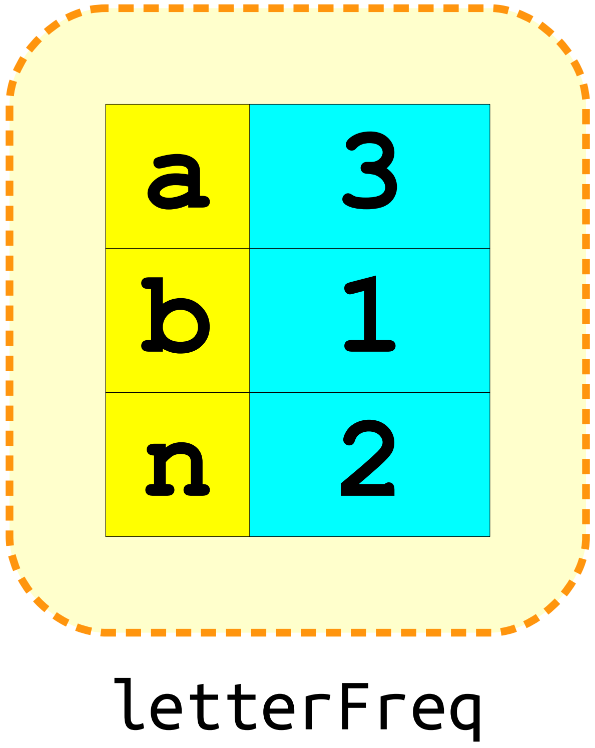


a	**3**
b	**1**
n	**2**

`letterFreq`

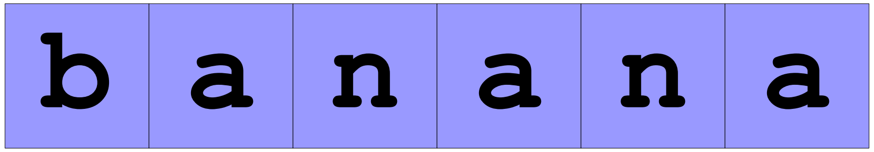
Counting Sort

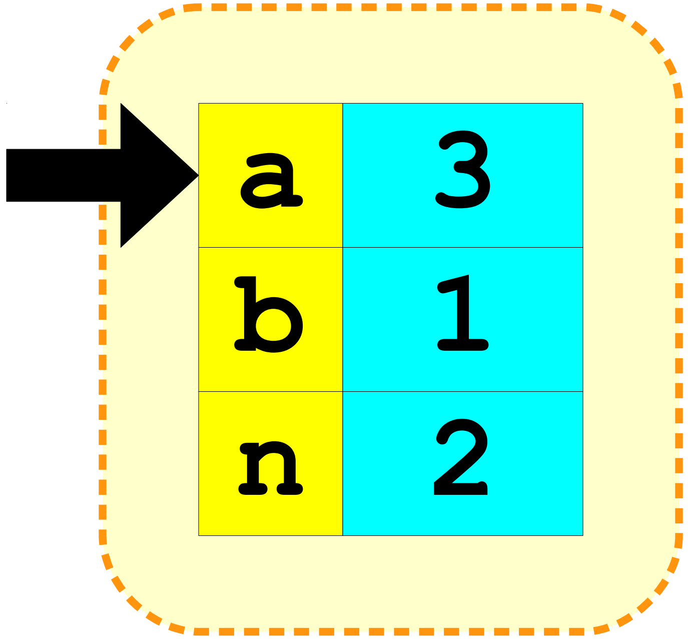




```
for (char ch: letterFreq) {
    for (int i = 0; i < letterFreq[ch]; i++) {
        result += ch;
    }
}
```

Counting Sort



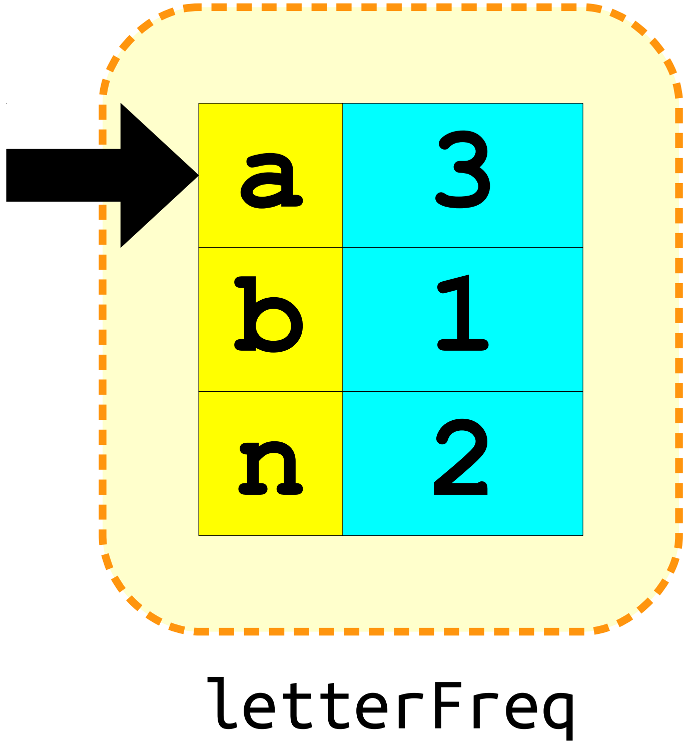


letterFreq

```
for (char ch: letterFreq) {
    for (int i = 0; i < letterFreq[ch]; i++) {
        result += ch;
    }
}
```

Counting Sort

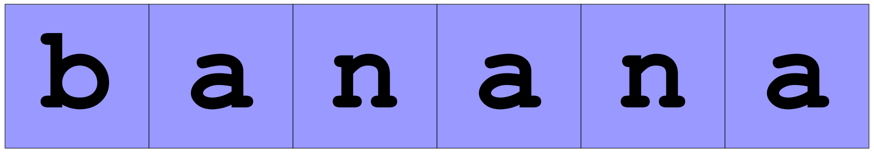
b	a	n	a	n	a

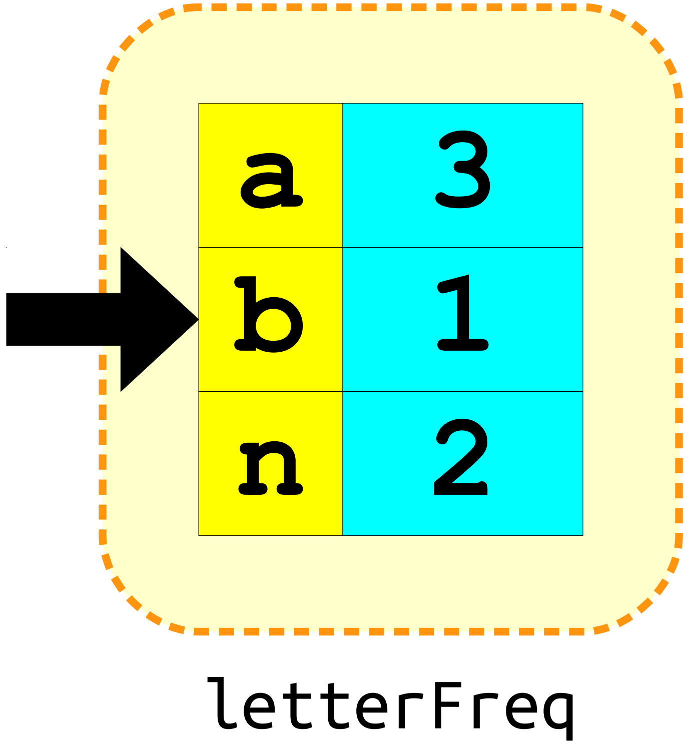


```
for (char ch: letterFreq) {
    for (int i = 0; i < letterFreq[ch]; i++) {
        result += ch;
    }
}
```

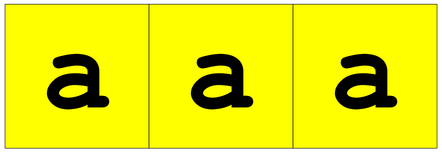
a	a	a

Counting Sort

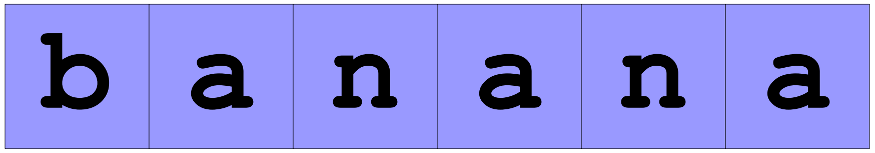


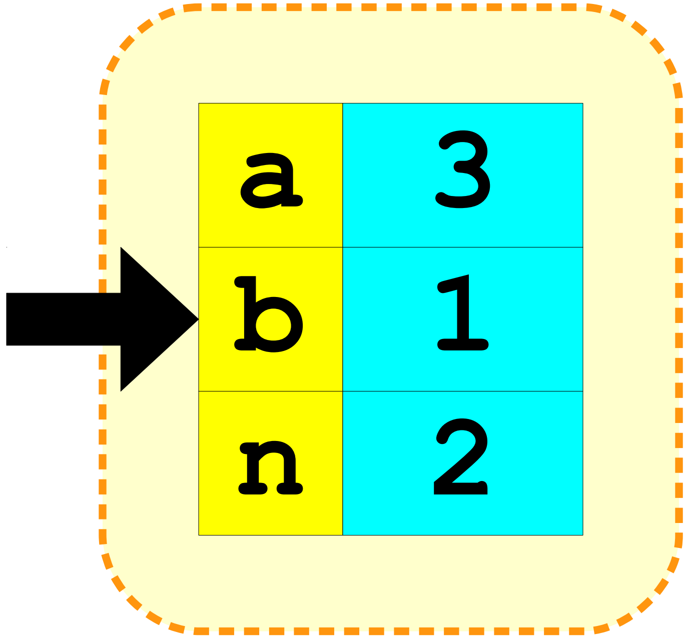


```
for (char ch: letterFreq) {
    for (int i = 0; i < letterFreq[ch]; i++) {
        result += ch;
    }
}
```



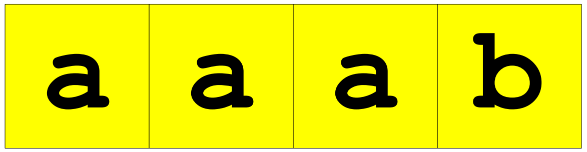
Counting Sort



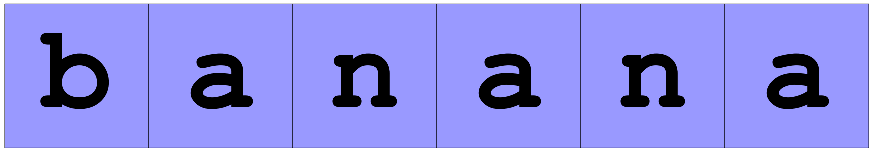


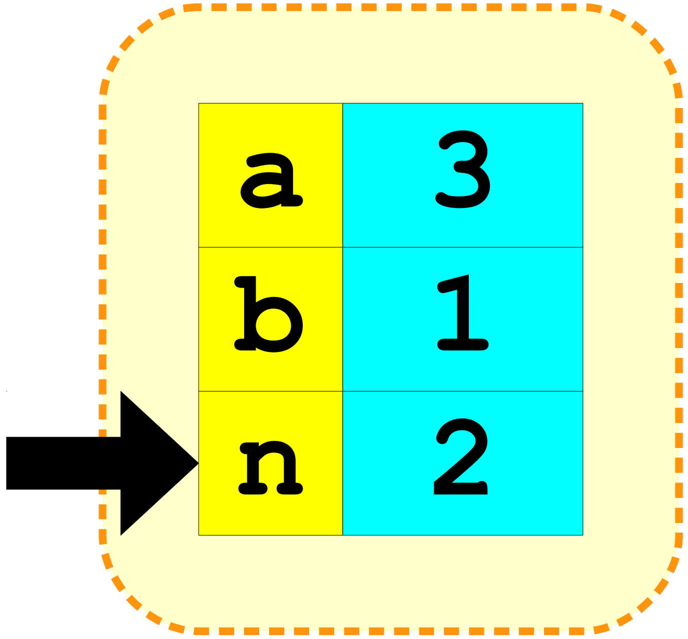
letterFreq

```
for (char ch: letterFreq) {
    for (int i = 0; i < letterFreq[ch]; i++) {
        result += ch;
    }
}
```



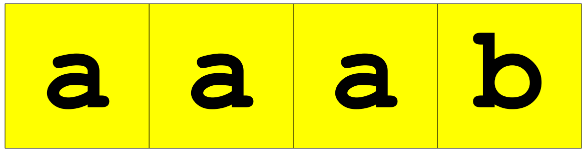
Counting Sort



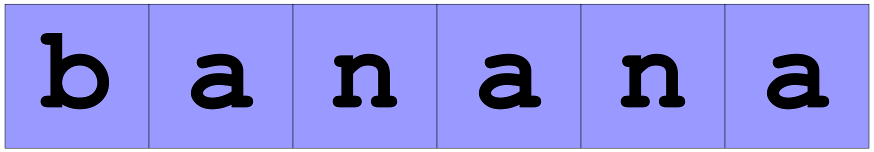


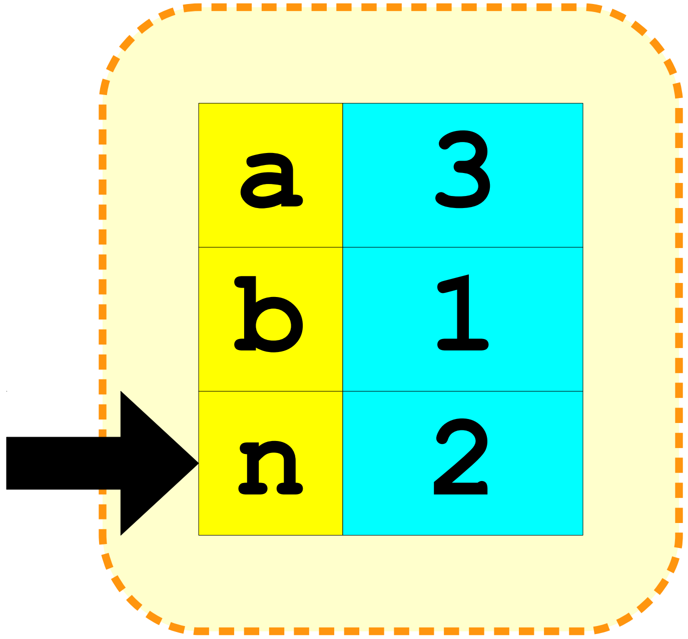
letterFreq

```
for (char ch: letterFreq) {
    for (int i = 0; i < letterFreq[ch]; i++) {
        result += ch;
    }
}
```



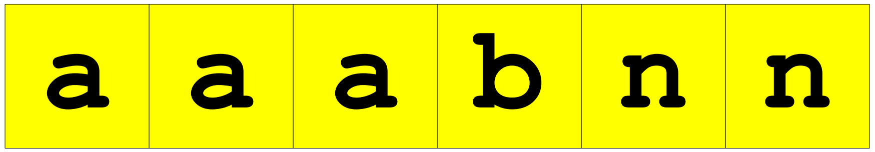
Counting Sort



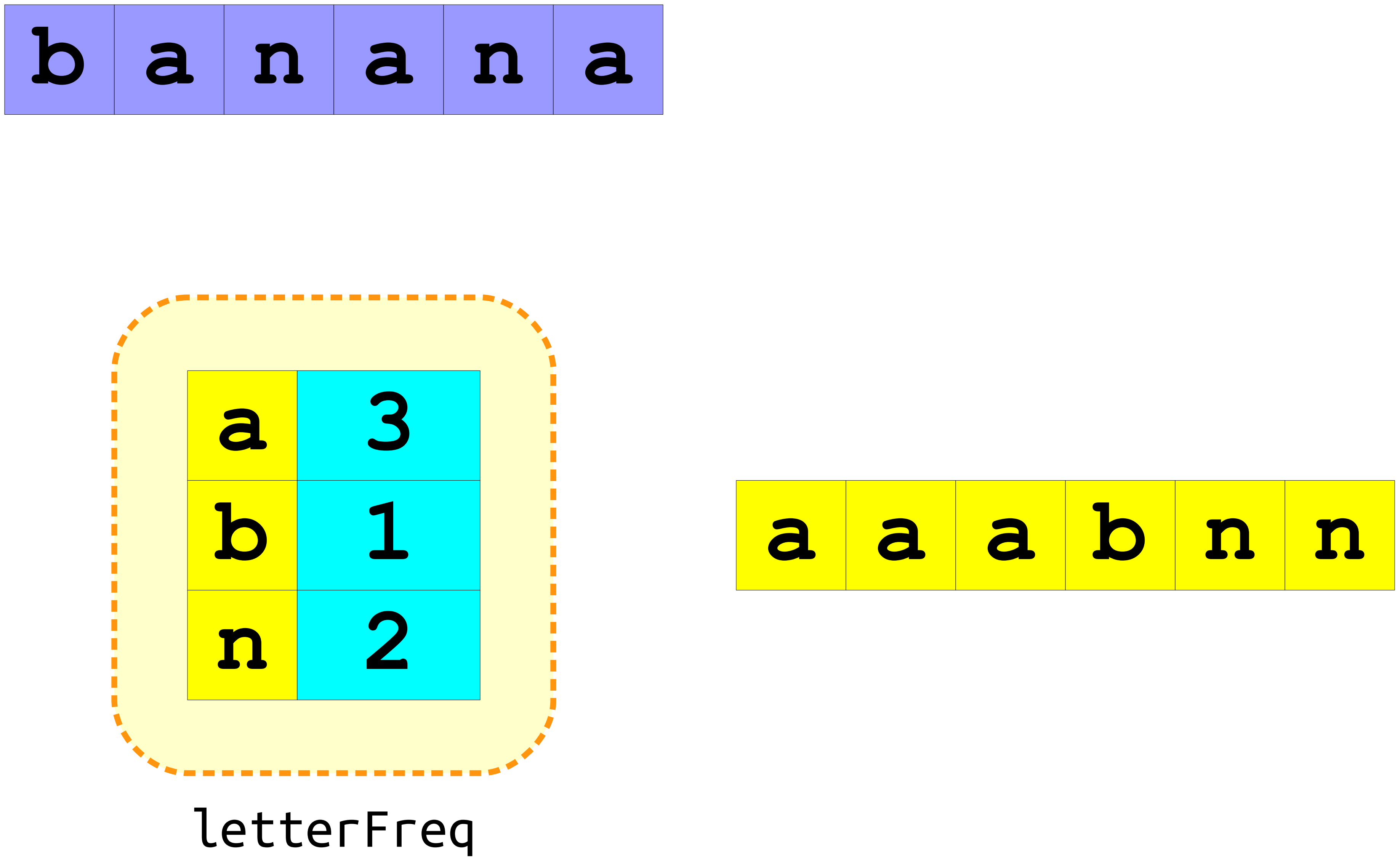


letterFreq

```
for (char ch: letterFreq) {
    for (int i = 0; i < letterFreq[ch]; i++) {
        result += ch;
    }
}
```



Counting Sort


b a n a n a
a 3
b 1
n 2
letterFreq
a a a b n n